

# On the probe complexities of membership and perfect hashing

Rasmus Pagh<sup>1</sup>  
September 9, 2000

## 1 Problems and notation

This paper considers the following static data structure problems.

**MEMBERSHIP**( $u, n, s$ ): Store a set  $S \subseteq [u]$ ,  $|S| = n \leq u/2$ , in a data structure  $\phi_m(S) \in \{0, 1\}^s$ . The objective is to accommodate efficient membership queries.

**PERFECT HASHING**( $u, n, r, s$ ): For a set  $S \subseteq [u]$ ,  $|S| = n$ , store a function  $f_S : [u] \rightarrow [r]$ ,  $r \geq n$ , with  $f_S$  1-1 on  $S$ , in a data structure  $\phi_{ph}(S) \in \{0, 1\}^s$ . The objective is to accommodate efficient function value queries.

We use the notation  $[x]$  for the set  $\{1, \dots, x\}$ . Two models for the query algorithm are considered. The *bitprobe* model of computation allows the query algorithm to probe single bits of the data structure. See the full version of [2] for details. The *cellprobe* model of computation, formally defined in [9], allows the query algorithm to probe “cells” of  $\log u$  consecutive bits.<sup>2</sup> When using the term “optimal”, it will be implied that we mean “to within a constant factor”, unless otherwise specified. Also, we assume that arguments to logarithms are always greater than or equal to 1, that is,  $\log(x)$  really means  $\log(\max(x, 1))$ .

### 1.1 Discussion

These above problems are, of course, intimately related. A data structure for **MEMBERSHIP** also encodes a perfect hash function (say, the rank of  $x$  among the elements of  $S$ ). Further, many such data structures have a unique cell of the data structure associated with each elements of  $S$  (not all, though, see [2]). In this case, the number of the cell may be used as a hash function value for elements in  $S$  (function values of other elements do not matter). Conversely, a  $k$  bitprobe ( $k$  cellprobe) solution to **PERFECT HASHING**( $u, n, r, s$ ) combined with a hash table gives a  $k + \log u$  bitprobe

---

<sup>1</sup>**BRICS**, University of Aarhus, Denmark. E-mail: pagh@brics.dk

<sup>2</sup>We will often implicitly assume that numbers which must be integers are rounded up or down appropriately.

$(k+1)$  cellprobe) solution to MEMBERSHIP( $u, n, s+r \log u$ ). If every function  $f_S$  has the property that each pre-image  $f_S^{-1}\{x\}$  has size at most  $R$  (we may have  $R \approx u/n$ ) and is “efficiently enumerable”, we even get a  $k + \log R$  bitprobe solution to MEMBERSHIP( $u, n, s + r \log R$ ) [6].

The combined problem, that is, a data structure which provides membership information as well as perfect hash function values, is referred to as DICTIONARY( $u, n, r, s$ ). The perfect hash function values can be used as indices to a table with a piece of *satellite information* for each element of  $S$  (this motivates the terminology). Often the satellite information is assumed to reside in the dictionary itself. However, in this case one can easily get a perfect hash function with range  $[n]$ , given that numbers in  $[n]$  can be satellite information. We prefer isolating the issue of satellite information, as this eliminates taking its size into account. However, one should be aware that DICTIONARY( $u, n, r, s$ ) requires an external table of size  $r$  to facilitate access to satellite information. As long as we are only interested in time and space bounds up to constant factors, we can consider MEMBERSHIP and PERFECT HASHING separately, and get solutions for DICTIONARY by simply “concatenating” the schemes.

## 1.2 Space bounds

The minimum number of bits for which it is possible encode a MEMBERSHIP data structure is  $s_m = \log \binom{u}{n} = \Theta(n \log(\frac{ue}{n}))$ . For PERFECT HASHING we consider just the case  $u = (1 + \Omega(1))r$ . Then the optimal space complexity is  $s_{ph} = \Theta(n^2/r + \log n + \log \log_r u)$  (nearly proved in [4, Thm. III.2.3.6]).

Bedre ref?

## 2 Generalized bitvectors

Bitvectors form a bitprobe optimal solution to MEMBERSHIP( $u, n, u$ ). The space usage of  $u$  bits is optimal in the case where  $n = \Omega(u)$ . If one insists on using minimal space, then  $\Omega(\log(u/n))$  bitprobes are needed [2]. An  $O(\log u)$  bitprobe scheme for MEMBERSHIP( $u, n, O(s_m)$ ) was devised by Brodnik and Munro [1]. It is optimal with respect to both space and bitprobes in the case  $n = u^{1-\Omega(1)}$ .

Here, we give a solution that uses optimal space and performs  $O(\log(u/n))$  bitprobes. Hence, it forms a “generalization” of bit vectors, that has optimal space and bitprobe complexity even for sparse sets. The data structure also allows PERFECT HASHING queries in  $O(\log(u/n))$  bitprobes.

The essential ingredient in our solution is a family of *bounded concentrators*, which are constant degree bipartite graphs with  $v$  vertices on the left,

$\theta v$  vertices on the right, for constant  $\theta < 1$ , having the property that any set  $V$  of no more than  $v/2$  left hand vertices can be matched to  $|V|$  vertices on the right hand side. A probabilistic argument shows that bounded concentrators of degree 6 with  $\theta = 2/3$  exist [7]. Simple, explicit constructions of bounded concentrators (with worse parameters) also exist [3, 5]. We denote left hand vertices by  $s_1, \dots, s_v$  and right hand vertices by  $t_1, \dots, t_{\theta v}$ , and assume that there is some ordering of the neighboring vertices of each vertex.

Check grad for  
hjre side

Our scheme is described recursively, as a step reducing the size of the universe considered followed by a scheme solving the a membership problem in the reduced universe. Each step requires the probe algorithm to read  $O(1)$  bits from a data structure of  $O(n)$  bits, and reduces the size of the universe by a constant factor. At the end of the recursion we have  $u = O(n)$ , and a bitvector can be used. The index looked up in the bitvector defines a perfect hash function (use any value for elements not in the set). This gives the following theorem.

**Theorem 1** *The bitprobe complexities of MEMBERSHIP( $u, n, s_m$ ) and PERFECT HASHING( $u, n, O(n), s_m$ ) are  $O(\log(u/n))$ .*

We have yet to describe the universe reduction step. If  $u \leq \frac{4n}{1-\theta}$ , we use a bitvector. Otherwise, split  $[u]$  into  $v = 2n$  parts  $U_1, \dots, U_{2n}$  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, vertices of  $V$  can be matched to a set  $W \subseteq \{t_1, \dots, t_{\theta v}\}$ . Suppose that  $s_i \in V$  is matched to its  $k$ th neighbor. Then we write the  $O(1)$  bit number  $k$  as entry  $i$  of a  $2n$ -element table  $T$ . Table entries vertices not in  $V$  are set to a special value. The query algorithm uses this as follows: On input  $x \in U_i$ , we check is  $s_i$  is in  $V$ . If not, then  $U_i \cap S = \emptyset$ , so  $x \notin S$ . Otherwise, if  $s_i$  is matched to  $t_j$ , recurse on the query  $j \lceil u/2n \rceil - z_x$ , where  $z_x$  is the number of  $x$  in some arbitrary numbering  $0, \dots, \lceil u/2n \rceil - 1$  of  $U_i$ . By the matching property, no query for another element of  $[u]$  translates into this query. Also, recursive queries are in the universe  $[\theta 2n \lceil u/2n \rceil]$ , which has size at most  $\frac{1+\theta}{2}u$ .

### 3 Bitprobe lower bounds for perfect hashing

#### 3.1 Minimal perfect hashing

A natural question to ask is whether theorem 1 can be extended to PERFECT HASHING( $u, n, n, O(s_m)$ ), i.e., to *minimal* perfect hashing. We answer this

question negatively, by showing that close to  $\log n$  bitprobes must be used in this case, even if the universe is only slightly larger than  $n$ .

**Lemma 2** *If PERFECT HASHING( $u, n, r, s$ ) has bitprobe complexity  $t < \log n$ , then  $u < n \binom{r}{2^t} / \binom{n-1}{2^t}$ .*

*Proof.* If  $t$  bitprobes suffice, each element  $x \in [u]$  can map to less than  $2^t$  different values in  $[r]$ . Now suppose  $u \geq n \binom{r}{2^t} / \binom{n-1}{2^t}$ . We want to argue that there then exists a set of  $n$  elements which can only map to  $n - 1$  values in  $[r]$  (contradicting the assumption that there is a 1-1 map on these elements). Indeed, each set of  $2^t$  elements lies within  $\binom{r-2^t}{n-1-2^t}$  sets of size  $n - 1$ , so  $n \binom{r}{n-1} / \binom{r-2^t}{n-1-2^t} = n \binom{r}{2^t} / \binom{n-1}{2^t}$  elements are enough to ensure the existence of the desired set.  $\square$

**Corollary 3** *For  $u = (1 + \Omega(1))r$ , the bitprobe complexity of PERFECT HASHING( $u, n, n, s$ ) is at least  $\log(n) - O(1)$ .*

*Proof.* Suppose  $t = \log(n) - c$  bitprobes suffice. Then by lemma 2,  $u < n r / (n - 2^t) = (1 - 2^{-c})^{-1} r$ . By the assumption on  $u$ , we must have  $c = O(1)$ .  $\square$

Of course,  $O(\log(u/n))$  bitprobes is a poor bound when  $u$  is large. Schmidt and Siegel [8] have given a PERFECT HASHING( $u, n, n, O(s_{\text{ph}})$ ) scheme with bitprobe complexity  $O(\log n + \log \log u)$ . The following lower bound implies that the probe complexity is optimal among schemes using space  $s = 2^{(n + \log u)^{1 - \Omega(1)}}$ , in the case  $r = n = u^{1 - \Omega(1)}$ .

**Proposition 4** *The bitprobe complexity of PERFECT HASHING( $u, n, r, s$ ) is greater than  $\log \log_{rs} u$ .*

*Proof.* Suppose at most  $t$  bit positions are probed. Then for each element  $x \in [u]$  there is a set of less than  $2^t$  bit positions which can be probed on input  $x$  (for some data structure). In particular, there is a set  $B$  of less than  $2^t$  bit positions which determine the hash function values of  $\lceil u/s^{2^t} \rceil$  elements. Since there cannot be two of these elements with hash function values depending on the bits of  $B$  in the same way, we must have  $r^{2^t} > u/s^{2^t}$ , yielding the desired result.  $\square$

## 3.2 Limited adaptivity

We now consider PERFECT HASHING query schemes with few or no *adaptive* probes. *Non-adaptive* probes are initial probes depending only on the input to the query algorithm, and not on results of other probes (they can be thought of as carried out in parallel). Some non-adaptive probes may even not depend on the input – these are referred to as *fixed*. We first consider query algorithms in which all probes are non-adaptive.

**Proposition 5** *There is a constant  $c$  such that for  $s = O(s_{\text{ph}})$  and  $u > c^s$ , PERFECT HASHING( $u, n, r, s$ ) has non-adaptive bitprobe complexity  $\Omega(s)$ .*

*Proof.* Suppose there is a non-adaptive scheme using  $t$  probes. On input  $x \in [u]$  the scheme reads bit positions  $B_x \subseteq [s]$  where  $|B_x| \leq t$ . Let  $H = \Omega(s_{\text{ph}})$  be the minimum number of bits needed to encode a perfect hash function with range  $[r]$  for  $n$  elements from a universe of size  $\lfloor \sqrt{u} \rfloor$ . It is sufficient to show that we must have  $t > H/2$ . Assume  $t \leq H/2$ . 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  $u \geq \lfloor \sqrt{u} \rfloor \binom{s}{t} / \binom{H-1}{t}$  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  $\cup_{x \in U'} B_x$  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 read if minimal space is to be used. This should be compared with the  $\lceil \log r \rceil$  bits of the computed result. By a simple reduction (looking at the problem of computing a single bit of the output of a perfect hash function) we obtain:

**Corollary 6** *For any  $s$  there is a decision problem where data structures of size  $s$  require  $\Omega(s/\log s)$  non-adaptive bitprobes, while  $O(\log s)$  adaptive bitprobes suffice.*

We conclude this section by studying the case of exactly *one* adaptive bitprobe. In the case where all other probes are fixed, we show that one needs either  $r > \log(e)n$  or  $\Omega(s_{\text{ph}})$  fixed probes, regardless of the size of the data structure. When non-adaptive probes are not necessarily fixed ...

## 4 Membership in the cell probe model

### 4.1 Impossibility of two-probe schemes

## 4.2 Scheme with one fixed and two parallel probes

### References

- [1] Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM J. Comput.*, 28(5):1627–1640 (electronic), 1999.
- [2] 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.
- [3] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. System Sci.*, 22(3):407–420, 1981. Special issued dedicated to Michael Machtay.
- [4] Kurt Mehlhorn. *Data structures and algorithms. 1, Sorting and searching*. Springer-Verlag, Berlin, 1984.
- [5] Moshe Morgenstern. Natural bounded concentrators. *Combinatorica*, 15(1):111–122, 1995.
- [6] 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.
- [7] Nicholas Pippenger. Superconcentrators. *SIAM J. Comput.*, 6(2):298–304, 1977.
- [8] Jeanette P. Schmidt and Alan Siegel. The spatial complexity of oblivious  $k$ -probe hash functions. *SIAM J. Comput.*, 19(5):775–786, 1990.
- [9] Andrew Chi-Chih Yao. Should tables be sorted? *J. Assoc. Comput. Mach.*, 28(3):615–628, 1981.