Explicit Deterministic Constructions for Membership in the Bitprobe Model

Jaikumar Radhakrishnan[1], Venkatesh Raman[2], S. Srinivasa Rao[2]

[1] Tata Institute of Fundamental Research, Mumbai.
jaikumar@tcs.tifr.res.in
[2] Institute of Mathematical Sciences, Chennai, India 600 113.
{vraman,ssrao}@imsc.ernet.in

Abstract. We look at time-space tradeoffs for the static membership problem in the bit-probe model. The problem is to represent a set of size up to n from a universe of size m using a small number of bits so that given an element of the universe, its membership in the set can be determined with as few bit probes to the representation as possible.
We show several deterministic upper bounds for the case when the number of bit probes, is small, by explicit constructions, culminating in one that uses $o(m)$ bits of space where membership can be determined with $\lceil \lg \lg n \rceil + 2$ adaptive bit probes. We also show two tight lower bounds on space for a restricted two probe adaptive scheme.

1 Introduction

We look at the static membership problem: Given a subset S of up to n keys drawn from a universe of size m, store it so that queries of the form "Is x in S?" can be answered quickly. We study this problem in the bit-probe model where space is counted as the number of bits used to store the data structure and time as the number of bits of the data structure looked at in answering a query.

A simple characteristic bit vector gives a solution to the problem using m bits of space in which membership queries can be answered using one bit probe. On the other hand, the structures given by Fredman et al.[4], Brodnik and Munro [1] and Pagh [5] can be used to get a scheme that uses $O(n \lg m)$ bits of space in which membership queries can be answered using $O(\lg m)$ bit probes. Recently Pagh [6] has given a structure that requires $O(s_{m,n})$ bits of space and supports membership queries using $O(\lg(m/n))$ bit probes to the structure, where $s_{m,n} = \Theta(n \lg(m/n))$ is the information theoretic lower bound on space for any structure storing an n element subset of an m element universe.

Buhrman et al.[2] have shown that both the above schemes are optimal. In particular they have shown that any deterministic scheme that answers membership queries using one bit probe requires at least m bits of space and any deterministic scheme using $O(s_{m,n})$ bits of space requires at least $\Omega(\lg(m/n))$ probes to answer membership queries. They have considered the intermediate ranges and have given some upper and lower bounds for randomized as well as deterministic versions. Their main result is that the optimal $O(n \lg m)$ bits (for

$n \leq m^{1-\Omega(1)}$) and one bit probe per query are sufficient, if the query algorithm is allowed to make errors (both sided) with a small probability. For the deterministic case, however, they have given some non-constructive upper bounds. They have also given some explicit structures for the case when t is large ($t \geq \lg n$).

Our main contribution in this paper, is some improved deterministic upper bounds for the problem using explicit constructions, particularly for small values of t. For sets of size at most 2, we give a scheme that uses $O(m^{2/3})$ bits of space and answers queries using 2 probes. This improves the $O(m^{3/4})$ bit scheme in [2] shown using probabilistic arguments. We also show that the space bound is optimal for a restricted two probe scheme. We then generalize this to a $\lceil \lg \lg n \rceil + 2$ probe scheme for storing sets of size at most n, which uses $o(m)$ bits of space. This is the best known constructive scheme (in terms of the number of bit probes used) for general n that uses $o(m)$ bits of space, though it is known [2] (using probabilistic arguments) that there exists a scheme using $o(m)$ bits of space where queries can be answered using a constant number of bit probes.

The next section introduces some definitions. The following section gives improved upper bounds for deterministic schemes. In section 4, we give some space lower bounds for a restricted class of two probe schemes, matching our upper bound. Finally, Section 5 concludes with some remarks and open problems.

2 Definitions

We reproduce the definition of a storing scheme, introduced in [2]. An (n, m, s)-storing scheme, is a method for representing any subset of size at most n over a universe of size m as an s-bit string. Formally, an (n, m, s)-storing scheme is a map ϕ from the subsets of size at most n of $\{1, 2, \ldots, m\}$ to $\{0, 1\}^s$. A deterministic (m, s, t)-query scheme is a family of m boolean decision trees $\{T_1, T_2, \ldots, T_m\}$, of depth at most t. Each internal node in a decision tree is marked with an index between 1 and s, indicating an address of a bit in an s-bit data structure. All the edges are labeled by "0" or "1" indicating the bit stored in the parent node. The leaf nodes are marked "Yes" or "No". Each tree T_i induces a map from $\{0, 1\}^s \rightarrow \{\text{Yes, No}\}$. An (n, m, s)-storing scheme and an (m, s, t)-query scheme T_i together form an (n, m, s, t)-scheme which solves the (n, m)-membership problem if $\forall S, x \ s.t. \ |S| \leq n, x \in U : T_x(\phi(S)) = Yes$ if and only if $x \in S$. A non-adaptive query scheme is a deterministic scheme where in each decision tree, all nodes on a particular level are marked with the same index.

We follow the convention that whenever the universe $\{1, \ldots, m\}$ is divided into blocks of size b (or m/b blocks), the elements $\{(i-1)b + 1, \ldots, ib\}$ from the universe belong to the ith block, for $1 \leq i \leq \lfloor m/b \rfloor$ and the remaining (at most b) elements belong to the last block. For integers x and a we define, $div(x, a) = \lfloor x/a \rfloor$ and $mod(x, a) = x - a \ div(x, a)$. To simplify the notation, we ignore integer rounding ups and downs at some places where they do not affect the asymptotic analysis.

3 Upper bounds for deterministic schemes

As observed in [2], the static dictionary structure given by Fredman, Komlos and Szemeredi [4] can be modified to give an adaptive (n, m, s, t)-scheme with $s = O(nkm^{1/k})$ and $t = O(\lg n + \lg\lg m) + k$, for any parameter $k \geq 1$. This gives a scheme when the number of probes is larger than $\lg n$. In this section, we look at schemes which require fewer number of probes albeit requiring more space.

For two element sets, Buhrman et al.[2] have given a non-adaptive scheme that uses $O(\sqrt{m})$ bits of space and answers queries using 3 probes. If the query scheme is adaptive, there is even a simpler structure. Our starting point is a generalization of this scheme for larger n.

Theorem 1. *There is an explicit adaptive (n, m, s, t)-scheme with $t = \lceil \lg(n+1) \rceil + 1$ and $s = (n + \lceil \lg(n+1) \rceil) m^{1/2}$.*

Proof. The structure consists of two parts. We divide the universe into blocks of size $m^{1/2}$. The first part consists of a table T of size $m^{1/2}$, each entry corresponding to a block. We call a block non-empty if at least one element from the given set falls into that block and empty otherwise. For each non-empty block, we store its rank (the number of non-empty blocks appearing before and including it) in the table entry of that block and store a string of zeroes for each empty block. Since the rank can be any number in the range $[1, \ldots, n]$ (and we store a zero for the empty blocks), we need $\lceil \lg(n+1) \rceil$ bits for storing each entry of the table T.

In the second part, we store the bit vectors corresponding to each non-empty block in the order in which they appear in the first part. For convenience, we call the jth bit vector as table T_j. Thus the total space required for the structure is at most $(n + \lceil \lg(n+1) \rceil) m^{1/2}$ bits.

Every element $x \in [m]$ is associated with $l+1$ locations, where l is the number of non-empty blocks: $t(x) = div(x, m^{1/2})$ in table T and $t_j(x) = mod(x, m^{1/2})$ in table T_j for $1 \leq j \leq l$. Given an element x, the query scheme first reads the entry j at location $t(x)$ in table T. If $j = 0$, the scheme answers 'No'. Otherwise it looks at the bit $T_j(t_j(x))$ in the second part and answers 'Yes' if and only if it is a one. □

If only two probes are allowed, Buhrman et al.[2] have shown that, any non-adaptive scheme must use m bits of space. For sets of size at most 2, they have also proved the existence of an adaptive scheme using 2 probes and $O(m^{3/4})$ bits of space. We improve it to the following:

Theorem 2. *There is an explicit adaptive scheme that stores sets of size at most 2 from a universe of size m using $O(m^{2/3})$ bits and answers queries using 2 bit-probes.*

Proof. Divide the universe into blocks of size $m^{1/3}$ each. There are $m^{2/3}$ blocks. Group $m^{1/3}$ consecutive blocks into a superblock. There are $m^{1/3}$ superblocks of size $m^{2/3}$ each.

The storage scheme consists of three tables T, T_0 and T_1, each of size $m^{2/3}$ bits. Each element $x \in [m]$ is associated with three locations, $t(x)$, $t_0(x)$ and $t_1(x)$, one in each of the three tables, as defined below. Let $b = m^{2/3}$ and $b_1 = m^{1/3}$. Then, $t(x) = div(x, b_1)$, $t_0(x) = mod(x, b)$ and $t_1(x) = div(x, b)\ b_1 + mod(x, b_1)$. Given an element $x \in [m]$, the query scheme first looks at $T(t(x))$. If $T(t(x)) = j$, it looks at $T_j(t_j(x))$ and answers 'Yes' if and only if it is 1, for $j \in \{0, 1\}$.

To represent a set $\{x, y\}$, if both the elements belong to the same superblock (i.e. if $div(x, b) = div(y, b)$), then we set the bits $T(t(x))$ and $T(t(y))$ to 0, all other bits in T to 1; $T_0(t_0(x))$ and $T_0(t_0(y))$ to 1 and all other bits in T_0 and T_1 to 0. In other words, we represent the characteristic vector of the superblock containing both the elements, in T_0, in this case.

Otherwise, if both the elements belong to different superblocks, we set $T(t(x))$, $T(t(y))$, $T_1(t_1(x))$ and $T_1(t_1(y))$ to 1 and all other bits in T, T_0 and T_1 to 0. In this case, each superblock has at most one non-empty block containing one element. So in T_1, for each superblock, we store the characteristic vector of the only non-empty block in it (if it exists) or any one block in it (which is a sequence of zeroes) otherwise. One can easily verify that the storage scheme is valid and that the query scheme answers membership queries correctly. □

One can immediately generalize this scheme for larger n to prove the following. Notice that the number of probes is slightly smaller than that used in Theorem 1, though the space used is larger.

Theorem 3. *There is an explicit adaptive (n, m, s, t)-scheme with $t = 1 + \lceil \lg(\lfloor n/2 \rfloor + 2) \rceil$ and $s = O(m^{2/3}(n/2 + \lg(n/2 + 2) + 1))$.*

Proof Sketch: The idea is to distinguish superblocks containing at least 2 elements from those containing at most one element.

In the first level, if a superblock contains at least 2 elements, we store its rank among all superblocks containing at least 2 elements, with all its blocks. Since there can be at most $\lfloor n/2 \rfloor$ superblocks containing at least 2 elements, the rank can be any number in the range $\{1, \ldots, \lfloor n/2 \rfloor\}$. For blocks which fall into superblocks containing at most one element, we store the number $\lfloor n/2 \rfloor + 1$, if the block is non-empty and a sequence of $\lceil \lg(\lfloor n/2 \rfloor + 2) \rceil$ zeroes, otherwise.

The second level consists of $\lfloor n/2 \rfloor + 1$ bit vectors of size $m^{2/3}$ each. We will store the characteristic vector of the jth superblock in the jth bit vector for $1 \leq j \leq l$, where l is the number of superblocks containing at least 2 elements. We will store all zeroes in the bit vectors numbered $l + 1$ to $\lfloor n/2 \rfloor$. In the $(\lfloor n/2 \rfloor + 1)$st bit vector, for each superblock we store the characteristic vector of the only non-empty block in it, if it has exactly one non-empty block or a sequence of zeroes otherwise.

On query x, we look at the first level entry of the block corresponding to x. We answer that the element is not present, if the entry is a sequence of zeroes. Otherwise, if it is a number k in the range $[1, \ldots, \lfloor n/2 \rfloor]$, we look at the corresponding location of x in the kth bit vector in the second level (which stores the bit vector corresponding to the superblock containing x). Otherwise (if the

number is $\lfloor n/2 \rfloor + 1$), we look at the corresponding location of x in the last bit vector and answer accordingly. □

This can be further generalized as follows. In the first level, we will distinguish the superblocks having at least k elements (for some integer k) from those with at most $k-1$ elements in them. For superblocks having at least k elements, we store the rank of that superblock among all such superblocks, in all the blocks of that superblock. For the other superblocks, we store the rank of the block among all non-empty blocks in that superblock, if the block is non-empty and a sequence of zeroes otherwise. The second level will have $\lfloor n/k \rfloor + k - 1$ bit vectors of length $m^{2/3}$ each where in the first $\lfloor n/k \rfloor$ bit vectors, we store the characteristic vectors of the at most $\lfloor n/k \rfloor$ superblocks containing at least k elements in them (in the order of increasing rank) and pad the rest of them with zeroes. Each of the $(\lfloor n/k \rfloor + j)$th bit vectors, for $1 \leq j \leq k-1$, stores one block for every superblock. This block is the jth non-empty block in that superblock, if that superblock contains at least j non-empty blocks and at most $k-1$ elements; we store a sequence of zeroes otherwise. The query scheme is straightforward. This results in the following.

Corollary 1. *There is an explicit adaptive (n, m, s, t)-scheme with $t = 1 + \lceil \lg(\lfloor n/k \rfloor + k) \rceil$ and $s = O(m^{2/3}(n/k + \lg(n/k + k) + k))$.*

Choosing $k = \lceil \sqrt{n} \rceil$, we get an explicit adaptive (n, m, s, t)-scheme with $t = 2 + \lceil \frac{1}{2} \lg n \rceil$ and $s = O(m^{2/3}\sqrt{n})$.

Actually, by choosing the block sizes to be $\frac{m^{1/3}(\lg n)^{2/3}}{n^{1/3}}$ and the sizes of the superblocks to be $\frac{m^{2/3}(\lg n)^{1/3}}{n^{1/6}}$ we get the following improved scheme:

Corollary 2. *There is an explicit adaptive (n, m, s, t)-scheme with $t = 2 + \lceil \frac{1}{2} \lg n \rceil$ and $s = O(m^{2/3}(n \lg n)^{1/3})$.*

We generalize this to the following:

Theorem 4. *There is an explicit adaptive (n, m, s, t)-scheme with $t = \lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil + 1$ and $s = m^{k/(k+1)} \left(\lg k + \frac{1}{k} \lg n + k n^{1/k} \right)$, for $k \geq 1$.*

Proof. We divide the universe into blocks of size b (to be determined later) and construct a complete b-ary tree with these blocks at the leaves. Let the height of this tree be k. Thus, we have $m = b^{k+1}$ or $b = m^{1/(k+1)}$. Given a set S of n elements from the universe, we store it using a three level structure. We call a block non-empty if at least one element of the given set S belongs to that block and call it empty otherwise. We define the height of a node in the tree to be the length of the path (the number of nodes in the path) from that node to any leaf in the subtree rooted at that node. Note that the height of the root is $k+1$ and that of any leaf is one.

In the first level we store an index in the range $[0, \ldots, k-1]$ corresponding to each block. Thus the first level consists of a table B of size b^k where each entry is a $\lceil \lg k \rceil$ bit number. The index stored for an empty block is 0. For a

non-empty block, we store the height $h \leq k-1$ of its ancestor (excluding the root and the first level nodes of the tree) x of maximum height such that the total number of elements falling into all the blocks in the subtree rooted at node x is more than $\lfloor n^{h/k} \rfloor$. This will be a number in the range $[0, \ldots, k-1]$.

In the second level we store a number in the range $[1, \ldots, \lceil n^{1/k} \rceil - 1]$ corresponding to each block. Thus this level consists of a table T of size b^k, each entry of which is a $\lceil \lg n^{1/k} \rceil$ bit number. The number stored for an empty block is 0. For a non-empty block, we store the following:

Observe that given any node x at height h which has at most $\lfloor n^{h/k} \rfloor$ elements from the set, the number of its children which have more than $\lfloor n^{(h-1)/k} \rfloor$ elements from the set is less than $\lceil n^{1/k} \rceil$. Suppose the index stored for a block is l. It means that the ancestor x of that block at height l has more than $\lfloor n^{l/k} \rfloor$ elements and the ancestor y at height $l+1$ has at most $\lfloor n^{(l+1)/k} \rfloor$ elements. Hence y can have less than $\lceil n^{1/k} \rceil$ children which have more than $\lfloor n^{l/k} \rfloor$ elements. Call these the 'large' children. With all the leaves rooted at each large child of y, we store the rank of that child among all large children (from left to right) in the second level.

In the third level, we have k tables, each of size $\lceil n^{1/k} \rceil m/b$ bits. The ith table stores the representations of all blocks whose first level entry (in table B) is i. We think of the ith table as a set of $\lceil n^{1/k} \rceil$ bit vectors, each of length m/b. Each of these bit vectors in the ith level stores the characteristic vector of a particular child for each node at height i of the tree, in the left to right order. For each block (of size b) with first level entry i and second level entry j, we store the characteristic vector of that block in the jth bit vector of the ith table at the location corresponding to its block of size b^{k-i}. We store zeroes (i.e. the characteristic vector of an empty block of appropriate size) at all other locations not specified above.

Every element $x \in [m]$ is associated with $k+2$ locations $b(x)$, $t(x)$ and $t_i(x)$ for $0 \leq i \leq k-1$, as defined below: $b(x) = t(x) = div(x, b)$, $t_i(x) = mod(div(x, b^{k-i})b^i + mod(x, b^i), b^k)$.

Given an element x, the query scheme first reads $i = B(b(x))$ and $j = T(t(x))$ from the first two levels of the structure. If $j = 0$, it answers 'No'. Otherwise, it reads the jth bit in the table entry at location $t_i(x)$ in table T_i and answers 'Yes' if and only if it is 1.

The space required for the structure is $s = b^k(\lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil + \frac{m}{b} k \lfloor n^{1/k} \rfloor)$ bits. Substituting $b = m^{1/(k+1)}$ makes the space complexity to be $m^{k/(k+1)}(\lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil + kn^{1/k})$. The number of probes required to answer a query is $t = \lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil + 1$. □

One can slightly improve the space complexity of the above structure by choosing non-uniform block sizes and making the block sizes (branching factors at each level, in the above tree structure) to be a function of n. More precisely, by choosing the branching factor of all the nodes at level i in the above tree structure to be b_i, where $b_i = m^{1-\frac{i}{k+1}} \left(\frac{\lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil}{\lceil n^{1/k} \rceil} \right)^{i/(k+1)}$, we get

Corollary 3. *There is an explicit adaptive (n, m, s, t)-scheme with $t = \lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil + 1$ and $s = (k+1)m^{k/(k+1)} \left(n(\lceil \lg k \rceil + \lceil \lg n^{1/k} \rceil)\right)^{1/(k+1)}$, for $k \geq 1$.*

By setting $k = \lg n$, we get

Corollary 4. *There is an explicit adaptive (n, m, s, t)-scheme with $t = \lceil \lg \lg n \rceil + 2$ and $s = o(m)$ when n is $O(m^{1/\lg \lg m})$.*

In the above adaptive scheme we first read $\lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil$ bits from the structure, and depending on these bits we look at one more bit in the next level to determine whether the query element is present. An obvious way to make this scheme non-adaptive is to read the $\lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil$ bits and all possible $k \lceil n^{1/k} \rceil$ bits (in the next level) and determine the membership accordingly. Thus we get an explicit non-adaptive (n, m, s, t)-scheme with $t = \lceil \lg k \rceil + \lceil \frac{1}{k} \lg n \rceil + k \lceil n^{1/k} \rceil$ and $s = tm^{k/(k+1)}$. By setting $k = \lceil \lg n \rceil$ in this, we get a non-adaptive scheme with $t = O(\lg n)$ and $s = o(m)$.

These schemes give the best known explicit adaptive and non-adaptive schemes respectively for general n using $o(m)$ bits.

4 Lower Bounds

Buhrman et al.[2] have shown that for any (n, m, s, t) scheme s is $\Omega(ntm^{1/t})$. One can achieve this bound easily for $n = 1$. They have also shown that for $n \geq 2$ any two probe non-adaptive scheme must use at least m bits of space. In this section, we show a space lower bound of $\Omega(m^{2/3})$ bits for a restricted class of adaptive schemes using two probes, for $n \geq 2$. Combining this with the upper bound of Theorem 2, this gives a tight lower bound for this class of restricted schemes. We conjecture that the lower bound applies even for unrestricted schemes. We also show a lower bound of $\Omega(m)$ bits for this restricted class of schemes for $n \geq 3$.

Any two-probe $O(s)$ bit adaptive scheme to represent sets of size at most 2 from a universe U of size m, can be assumed to satisfy the following conditions (without loss of generality):

1. It has three tables A, B and C each of size s bits.
2. Each $x \in U$ is associated with three locations $a(x)$, $b(x)$ and $c(x)$.
3. On query x, the query scheme first looks at $A(a(x))$. If $A(a(x)) = 0$ then it answers 'Yes' if and only if $B(b(x)) = 1$ else if $A(a(x)) = 1$ then it answers 'Yes' if and only if $C(c(x)) = 1$.
4. Let $A_i = \{x \in [m] : a(x) = i\}$, $B_i = \{b(x) : x \in A_i\}$ and $C_i = \{c(x) : x \in A_i\}$ for $1 \leq i \leq s$. For all $1 \leq i \leq s$, $|B_i| = |A_i|$ or $|A_i| = |C_i|$. I.e. the set of elements looking at a particular location in table A will all look at a distinct locations in one of the tables, B and C. (Otherwise, let $x, y, x', y' \in A_i$, $x \neq y$ and $x' \neq y'$ be such that $b(x) = b(y)$ and $c(x') = c(y')$. Then we can not represent the set $\{x, x'\}$.)

5. Each location of A, B and C is looked at by at least two elements of the universe, unless $s \geq m$. (If a location is looked at by only one element, then set that location to 1 or 0 depending on whether the corresponding element is present or not; we can remove that location and the element out of our scheme.)
6. There are at most two ones in B and C put together.

Define the following restrictions:

- R1. For $x, y \in [m], x \neq y$, $a(x) = a(y) \Rightarrow b(x) \neq b(y)$ and $c(x) \neq c(y)$.
- R2. For $i, j \in [s], i \neq j$, $B_i \cap B_j \neq \phi \Rightarrow C_i \cap C_j = \phi$.
- R3. Either B or C is all zeroes.

We show that if an adaptive $(2, m, s, 2)$ scheme satisfies R3 (or equivalently R1 and R2, as we will show), then s is $\Omega(m^{2/3})$. Note that the scheme given in Theorem 2 satisfies all these three conditions. We then show that if an adaptive $(n, m, s, 2)$ scheme for $n \geq 3$ satisfies R3, then $s \geq m$.

Theorem 5. *If an adaptive $(2, m, s, 2)$ scheme satisfies condition R3, then s is $\Omega(m^{2/3})$.*

Proof. We first show that (R3 $\Rightarrow$ R1 and R2) and then show that (R1 and R2 $\Rightarrow$ s is $\Omega(m^{2/3})$).

Let $a(x) = a(y)$ and $b(x) = b(y)$ for $x, y \in [m], x \neq y$. Consider an element $z \neq x$ such that $c(x) = c(z)$ (such an element exists by condition 5 above). Now, the set $\{y, z\}$ cannot be represented satisfying R3. Thus we have, R3 $\Rightarrow$ R1.

Again, let $a(x_1) = a(x_2) = i$, $a(y_1) = a(y_2) = j$, $b(x_1) = b(y_1)$ and $c(x_2) = c(y_2)$ (so that R2 is violated). Then, the set $\{x_2, y_1\}$ cannot be represented satisfying R3. Thus we have, R3 $\Rightarrow$ R2.

Observe that R1 implies

$$|A_i| = |B_i| = |C_i|, \ \forall i, 1 \leq i \leq s. \tag{1}$$

Hence

$$\sum_{i=1}^{s} |B_i| = \sum_{i=1}^{s} |A_i| = m. \tag{2}$$

By R2, the sets $B_i \times C_i$ are disjoint (no pair occurs in two of these Cartesian products). Thus, by Equation (1), $\sum_{i=1}^{s} |B_i|^2 \leq s^2$. By Cauchy-Schwarz, $s(\sum_{i=1}^{s} |B_i|/s)^2 \leq \sum_{i=1}^{s} |B_i|^2 \leq s^2$. By Equation (2), $\sum_i |B_i| = m$. Thus, $m^2/s \leq s^2$ *or* $s \geq m^{2/3}$. □

Remark: We observe that, in fact the condition R3 is equivalent to R1 and R2. To show this, it is enough to prove that R1 and R2 $\Rightarrow$ R3. We argue that any scheme that satisfies R1 and R2 can be converted into a scheme that satisfies R3 also.

Consider any scheme which satisfies R1 and R2 but not R3. So, there exists a set $\{x, y\}$ such that $a(x) \neq a(y)$ for which the scheme stores this set as follows (without loss of generality): $A(a(x)) = 0, A(a(y)) = 1, B(b(x)) = 1, C(c(y)) = 1$, $A(a(z)) = 1$ for all z for which $b(z) = b(x)$, $A(a(z)) = 0$ for all z for which $c(z) = c(y)$ and all other locations as zeroes.

Let $a(x) = i$ and $a(y) = j$. If $B_i \cap B_j = \phi$ then we can store this set as follows: $A(a(x)) = A(a(y)) = 0, B(b(x)) = B(b(y)) = 1$ and all other entries in A as 1s, and all entries in B and C as zeroes, satisfying R3. Condition R1 (and the fact that $B_i \cap B_j = \phi$) ensures that this is a valid scheme to represent the set $\{x, y\}$.

If $B_i \cap B_j \neq \phi$, then R2 ensures that $C_i \cap C_j = \phi$. In this case, to store the set $\{x, y\}$ we can set $A(a(x)) = A(a(y)) = 1, C(c(x)) = C(c(y)) = 1$ and all other entries as zeroes, satisfying R3.

We now show the following.

Theorem 6. *If an adaptive $(n, m, s, 2)$ scheme, for $n \geq 3$ satisfies condition R3, then $s \geq m$.*

Proof. We first observe that any two probe adaptive scheme satisfies conditions 1 to 5 of the adaptive schemes for sets of size at most 2. Consider an adaptive $(3, m, s, 2)$ scheme with $s < m$. One can find five elements x, y, y', z and z' from the universe such that $a(y) = a(y')$, $a(z) = a(z')$, $b(x) = b(y)$ and $c(x) = c(z)$. (Start by fixing x, y, z and then fix x' and y'.) Existence of such a situation is guaranteed by condition 5, as $s < m$. Then we can not represent the set $\{x, y', z'\}$ satisfying R3, contradicting the assumption. Hence, $s \geq m$. □

5 Conclusions

We have given several deterministic explicit schemes for the membership problem in the bit probe model for small values of t. Our main goal is to achieve $o(m)$ bit space and answer queries using as few probes as possible. We could achieve $\lceil \lg \lg n \rceil + 2$ adaptive probes through an explicit scheme, though it is known (probabilistically) that one can get a $o(m)$ bit structure which uses only 5 probes to answer queries. It is a challenging open problem to come up with explicit scheme achieving this bound. We conjecture that one can not get a three probe $o(m)$ bit structure.

One can also fix some space bound and ask for the least number of probes required to answer the queries. For example, if $s = O(n\sqrt{m})$, Theorem 1 gives a $\lg(n + 1) + 1$ probe adaptive scheme. It would be interesting to see if this can be improved. Also this scheme immediately gives an $n + O(\lg n)$ probe non-adaptive scheme, with the same space bound. Demaine et al.[3] have improved this to an $O(\sqrt{n \lg n})$ probe non-adaptive scheme with $s = O(\sqrt{mn \lg n})$.

Acknowledgment. Part of the work was done while the second author was visiting the University of Waterloo, Canada. He thanks Ian Munro and Erik Demaine for useful discussions.

References

1. A. Brodnik and J. I. Munro, "Membership in constant time and almost minimum space", *SIAM Journal on Computing*, **28(5)**, 1628-1640 (1999).
2. H. Buhrman, P. B. Miltersen, J. Radhakrishnan and S. Venkatesh, "Are Bitvectors Optimal?", *Proceedings of Symposium on Theory of Computing* (2000) 449-458.
3. E. D. Demaine, J. I. Munro, V. Raman and S. S. Rao, "Beating Bitvectors with Oblivious Bitprobes", *I.M.Sc. Technical Report* (2001).
4. M. L. Fredman, J. Komlós and E. Szemerédi, "Storing a sparse table with $O(1)$ access time", *Journal of the Association for Computing Machinery*, **31** (1984) 538-544.
5. Rasmus Pagh, "Low redundancy in dictionaries with O(1) worst case lookup time", *Proceedings of the International Colloquium on Automata, Languages and Programming, LNCS* **1644** (1999) 595-604.
6. Rasmus Pagh, "On the Cell Probe Complexity of Membership and Perfect Hashing", Proceedings of Symposium on Theory of Computing (2001).