Abstract

Title of thesis	The Bit Probe Model for Membership Queries: Non-Adaptive Bit Queries
	Ryan Blue, Master of Science, 2009
Thesis directed by	Professor William Gasarch Department of Computer Science

A common problem in computer science is how to efficiently store sets: when given a set, how do you store it so that you do not use much space and membership queries can be done quickly? One popular method is to use bit vectors. We use a model data structure that is a variant of bit vectors, the bit probe data structure, to demonstrate lower and upper bounds for bit vectors and similar structures. This thesis presents a survey of known results from literature, contributes some new proofs that have not appeared before, and gives complete proofs of known results that are missing from literature.

The Bit Probe Model for Membership Queries: Non-Adaptive Bit Queries

by

Ryan Blue

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2009

Contents

List of Tables

List of Figures

1 Introduction

Consider the following common problem: You want to store a set so that you do not use too much space, and you can answer membership questions easily. In practice, such a set could "contain" any object, but we can restrict our scope to sets that store just numbers because structures generally store just integers that are references, hash keys, or some other numeric representative. We can also assume that the numbers we will store are limited by some maximum value, for practicality or convenience. For instance, most integer representations are 32 or 64 bits and are more than sufficient for most applications.

The key concerns when storing such sets are space and number of queries. Two straightforward methods for storing sets are good demonstrations of the trade-offs involved, (assume numbers are limited by some maximum called U):

1. Store each member of the set in a sorted-order list of length n.
   - Probes needed to determine membership: $O(\log n)$ queries using binary search.
   - Space needed to store: n cells of $O(\log U)$ bits each.
2. Construct a bit vector with one bit for each possible value $(1 \ldots U)$. Store a 1 for each item in the set and a 0 otherwise.
   - Probes needed to determine membership: 1.
   - Space needed to store: U bits

Method 1 uses much less storage space, $O(n \cdot \log U)$, and uses an adaptive query strategy—where the item queried can depend on the previous queries.

Method 2 uses much more space, but achieves a constant number of 1 bit queries rather than $O(\log n)$ queries, each $\log U$ bits. Can we do better? Is there a way to achieve $O(1)$ bit queries and still use only space considerably less than U bits[1]?

This thesis focuses on non-adaptive variants of the bit vector approach, generalized as the bit probe data structure. The bit probe data structure is similar to bloom filters[4], however, the data structures we present are deterministic and must provide correct results.

In this thesis, We will survey the literature, include some new, simpler proofs that have not appeared before, and will give complete proofs of known results that are missing from literature. We will pay particular attention to real constants and contribute concrete space comparisons between data structures.

We now proceed formally and define the bit probe model. We only define and work with the non-adaptive case where all of the queries are made at the same time. Normally, we would use the phrase "nonadaptive" but will omit it for this reason.

Note 1.1 We use $[x]$ to denote the set $\{1, 2, \ldots, x\}$, where $x \in \mathbb{N}$.

Def 1.2 A $(U, n; s, q)$ *bit probe data structure for membership*, henceforth BPDS, consists of the following:

1. A function that will, given $A \subseteq [U]$ of size $\leq n$, output a vector $CELL \in \{0,1\}^s$.
2. A membership algorithm, MEM, consisting of two functions:

[1]We will show in Section 2 that the answer is YES.

(a) A function that takes as input $u \in [U]$ and outputs $(i_1, \ldots, i_q)$ where $1 \le i_1, \ldots, i_q \le s$. The intuition is that on input u we ask the bit queries $CELL[i_1], \cdots, CELL[i_q]$. When $q = 2$ we will use (a_u, b_u).

(b) A function that takes input $u \in [U]$ and outputs a boolean function f_u on q boolean variables. We require that

$$u \in U \text{ iff } f_u(CELL[i_1], \ldots, CELL[i_q]) = IN.$$

(We use IN for TRUE and OUT for FALSE since the real info is IN and OUT.)

$(U, n; s, q)$ are the parameters of the data structure, where:

- U is the size of the *universe*
- n is the size of sets we will be storing
- s is the number of bits in the data structure, the *space* used
- q is the number of bit *queries* in a membership test

Note 1.3

1. The universe is U, which is why we used U above, so any integer stored in the data structure is in the set $[U]$. We use U as a parameter here because many of the bounds will be in terms of U, like the examples above. The data structure stores some subset, A, of $[U]$ that is of size $\le n$, so membership questions have the form "$u \in A$?"

2. The function and process in Definition 1.2 need not be computable. Hence our lower bounds will be very strong. Our upper bounds will use easily computable functions.

3. The membership algorithm in this definition was nonadaptive: questions asked could not depend on prior answers given.

4. We will often use the notation s and q rather than $s(n, U)$ and $q(n, U)$, even though values of s and q will often depend on n and U.

5. In the motivating examples, (1) did not fit this model. (2) did fit this model and is a $(U, n; U, 1)$-BPDS (parameters $s = U$ and $q = 1$).

6. $CELL[i]$ need not be related to $i \in U$ in any way shape or form.

Note 1.4 In proving lower bounds we must guard against the data structure designer having a clever idea. We give two examples of what she might do. Its not clear that these ideas are clever or helpful; however, they are possibilities that our lower bounds must take into account.

1. Assume that we have a $(U, n; s, 2)$-BPDS. Assume that to make the membership query "$2 \in A$?" we need to make the bit queries $CELL[12]$ and $CELL[84]$. Given a number, you know which queries are asked, but how those bits are set may depend on which set was stored. For instance, the following is possible:

   (a) If the set $\{1, 2, 7\}$ is stored then $CELL[12] = 1$ and $CELL[84] = 0$.

   (b) If the set $\{2, 4, 9\}$ is stored the $CELL[12] = 0$ and $CELL[84] = 1$.

In this case it may be that the membership query algorithm on the question "$2 \in A$?" might tell you to, after making the bit-probe queries $CELL[12]$ and $CELL[84]$, XOR the bits together to get the answer to the membership query.

2. The following is possible:

   (a) If the membership query "$17 \in A$?" is made then the bit probe queries asked are $CELL[4]$ and $CELL[8]$.

   (b) If the membership query "$25 \in A$?" is made then the bit probe queries asked are $CELL[4]$ and $CELL[8]$.

   How could this be? Well, it could be that

$$17 \in A \text{ iff } CELL[4] = 1 \text{ xor } CELL[8] = 0$$

   and

$$25 \in A \text{ iff } CELL[4] = 0$$

   In this case here is how the types of sets are stored: (1) If $17, 25 \notin A$ then set $CELL[4] = 1$ and $CELL[8] = 1$. (2) If $17 \in A$ and $25 \notin A$ then set $CELL[4] = 1$ and $CELL[8] = 0$. (3) If $17 \notin A$ and $25 \in A$ then set $CELL[4] = 0$ and $CELL[8] = 0$. (4) If $17, 25 \in A$ then set $CELL[4] = 0$ and $CELL[8] = 1$. The point is that you can have two different membership queries use the same two bit probe queries.

Note 1.5 This definition puts no restrictions on the function in 2.(a) and this is a known problem with the bit probe data structure model. The model counts queries, but ignores how to determine which queries to make. This fact means that the upper bounds on bit probe data structures that are non-constructive, such as the bounds in Sections 5 and 6, cannot necessarily be achieved if the function requires being stored as a lookup table—such a table would contain $O(U)$ entries. The lower bounds, however, are stronger for this reason. We proceed despite this flaw.

Using this BPDS definition, Sections 2, 3, and 4 present "easy" bounds and methods to introduce bit probe data structures: Section 2 gives an easy data structure for our first non-trivial upper bound. Section 3 details a similarly simple lower bound. Section 4 defines probabilistic bit probe data structures and presents an adaptation of the easy upper bound to demonstrate the concept. Next, Sections 5 and 6 present upper bounds from literature, while Sections 7, 8, and 9 will give lower bounds from literature. We conclude in Section 10.

The results in Sections 2, 3, and 4 are new. Section 2 is the best constructive upper bound for $q \geq 5$. The upper bounds from literature in Sections 5 and 6 are given in a more complete form than in literature and contain a correction to the bound in Section 6. We also contribute complete proofs of the lower bounds in Sections 7, 8, and 9, as well as a new generalization of the lower bound in Section 8. We present all proofs from literature consistently using bipartite graphs rather than with hypergraphs or other structures. All of our analyses contribute careful calculations of constant terms, new tables of real space requirements, and new comparisons between all of the upper and lower bounds.

2 Easy Upper Bound: There is a $(U, n; c_o U^\delta, q)$-BPDS where $\delta = \frac{1}{\lceil q/n \rceil}$

In this section, we present a simple data structure that achieves $O(1)$ queries and uses space $\ll U$. This algorithm was devised as an understandable introduction to bit probe data structures, which is why we present it first here.

This algorithm, which is the product of correspondence between William Gasarch and Peter Bro Milterson, has not yet appeared in literature. Our presentation includes an analysis of the algorithm's constant factors, as well as a table of the space required for storage for sample values of n and q.

Each value is stored by setting q bits in the structure that may overlap. Using polynomials, it ensures that at least one bit for each stored value will not overlap the bits set to store other values. The key to this data structure is that if two polynomials of degree d pass through the same $d+1$ points, they *must* be the same polynomial.

2.1 The Upper Bound

Lemma 2.1 *Assume there is a map from $[U]$ to $\binom{[s]}{q}$ (q-sized subsets of $[s]$). We will denote the set u maps to by B_u. If, for all n-sized subsets $\{u_1, u_2, \ldots, u_n\} \subseteq [U]$ and any $u \in [U]$*

$$|B_u \cap \bigcup_{i=1}^{n} B_{u_i}| < q$$

Then there is a $(U, n; s, q))$-BPDS.

Proof:

To show Lemma 2.1, we show how to construct a BPDS.

Setting up the Data Structure:

- Initially all of the cells are set to 0.
- Let $A \subseteq [U]$, $|A| \leq n$.
- For each $u \in A$ set the bits of B_u to 1.

Making a Query:

- To ask "$u \in A$?", you make q probes to the bits specified by B_u.
- If all q are 1 then output YES else NO.

Why does this work?

- Clearly, if $u \in A$ then the answer returned will be YES.
- Let $A = \{u_1, \ldots, u_n\}$. Let $u \notin \{u_1, \ldots, u_n\}$. We need to show that if A is stored and "$u \in A$?" is asked then the answer will be NO.
- By assumption, we know that

  $$|B_u \cap \bigcup_{i=1}^{n} B_{u_i}| < q$$

  This means that B_u overlaps the n sets of locations that were set to 1 by at most $q-1$ cells and at least one cell will be set to 0. Therefore, the membership algorithm will correctly conclude that u is not in A.

▮

Lemma 2.1 assumes a map from $[U]$ to $\binom{[s]}{q}$. The proof of the theorem proceeds by giving such a map and showing that it is efficient.

Theorem 2.2 *Let $n, q \in \mathbb{N}$. Let $\delta = \frac{1}{\lceil q/n \rceil}$. There exists constants U_0, c_0 such that, for all $U \geq U_0$, there is a $(U, n; c_o U^\delta, q)$-BPDS.*

Proof:

Let d be a quantity to be named later. Let $\delta = 1/(d+1)$. Let p be a prime number such that $U^\delta \leq p \leq 2U^\delta$. Let $TUPLE$ be an injection of $[U]$ to $[U^\delta] \times \cdots \times [U^\delta]$ ($d+1$ times).

If $TUPLE(u) = (a_d, \ldots, a_0)$ then let

$$f_u(x) = a_d x^d + \cdots + a_0$$

Let

$$B_u = \{(1, f_u(1)), \ldots, (q, f_u(q))\}$$

Note that for any $w \in U (w \neq u)$,

$$|B_u \cap B_w| \leq d$$

This is because B_u and B_w are each made up of $q > d$ points on degree d polynomials. If B_u and B_w had $d+1$ points in common, then the polynomials must be the same. Because $TUPLE$ is an injection, u and w must be the same, which would be a contradiction.

Therefore, B_u can only overlap n other such sets in at most dn points.

$$|B_u \cap \bigcup_{i=1}^{n} B_{u_i}| \leq dn$$

We need $dn < q$ in order to apply Lemma 2.1, so we choose $d = \lceil q/n \rceil - 1$ to ensure this condition. Note that $\delta = \frac{1}{d+1} = \frac{1}{\lceil q/n \rceil}$. ▮

2.2 Actual Values

We now present a careful analysis of the actual bounds and constant factors for the easy upper bound.

Let p be the smallest prime larger than $U^{\frac{1}{\lceil q/n \rceil}}$.

Let $TUPLE$ be an injection from $[U]$ to $\mathbb{Z}_p \times \mathbb{Z}_p \times \ldots \times \mathbb{Z}_p$ ($d + 1$ times).

We think of $TUPLE$ as an injection from $[U]$ to polynomials over $\mathbb{Z}_p$ of degree d, so $TUPLE(u) = f_u(x)$. Next, the algorithm forms the set $B_u = \{(1, f_u(1)), \ldots, (q, f_u(q))\}$ for each $u \in A$. The question is: how efficiently can we store these sets?

Theorem 2.3 *The sets $B_u : u \in [U]$ can be stored in $s = q \cdot p$ bits.*

We show this theorem by way of the following lemmas:

Lemma 2.4 *The sets $B_u : u \in [U]$ must be stored in $s \geq q \cdot p$ bits.*

Proof: Consider a q by p bit table in which the set B_u is stored by setting the $f_u(1)$-th bit of the first row, the $f_u(2)$-th bit of the second row, and so on.

We claim that this scheme minimizes the number of bits required to store B_u, because there are $q \cdot p$ possible values to store, and any fewer bits $(q \cdot p - 1)$ would cause a collision somewhere in the table and a mistake on at least one decision. ∎

Lemma 2.5 *The sets $B_u : u \in [U]$ can be stored in $s \leq q \cdot p$ bits.*

Proof: The fact that each B_u is constructed via a degree d polynomial guarantees (see Section 2) that it differs in at least one bit from all the other sets. Hence, all of the B_u sets can be stored in the same $q \cdot p$ bits.

∎

Baker, Harman, and Pintz [3] showed that, for almost all c, there is a prime between c and $c + c^{0.525}$ (see [5] for a survey of this type of result). Therefore, p is bounded by $U^{\frac{1}{\lceil q/n \rceil}} + U^{\frac{0.525}{\lceil q/n \rceil}}$

Using this, all of the sets can be stored in

$$s = q \cdot p \leq q \cdot \left(U^{\frac{1}{\lceil q/n \rceil}} + U^{\frac{0.525}{\lceil q/n \rceil}} \right) bits$$

To get an idea of what this equation means, see Tables 1 and 2, which list values of s for different q and n. We use $U = 2^{32} - 1$, as an example universe, which corresponds to common 32 bit integers. We will come back to these space values later, when comparing this upper bound to those presented in Sections 5 and 6.

Note that values in each row have a pattern: a large decrease followed by slow increase and then another large decrease... This pattern is due to the $\lceil q/n \rceil$

$n \setminus q$	3	4	5	6
2	197619	263492	8370	10044
3	12885244197	263492	329365	395238
4	12885244197	17180325596	329365	395238
5	12885244197	17180325596	21475406995	395238
6	12885244197	17180325596	21475406995	25770488394
7	12885244197	17180325596	21475406995	25770488394
8	12885244197	17180325596	21475406995	25770488394
9	12885244197	17180325596	21475406995	25770488394
10	12885244197	17180325596	21475406995	25770488394

Table 1: Values of s for different q (3-6) and n. ($U = 2^{32} - 1$.)

$n \setminus q$	7	8	9	10
2	1918	2192	846	940
3	11718	13392	15066	2740
4	461111	526984	15066	16740
5	461111	526984	592857	658730
6	461111	526984	592857	658730
7	30065569793	526984	592857	658730
8	30065569793	34360651192	592857	658730
9	30065569793	34360651192	38655732591	658730
10	30065569793	34360651192	38655732591	42950813990

Table 2: Values of s for different q (7-10) and n. ($U = 2^{32} - 1$.)

term. The jumps are when this reaches a new integer and the slow increase corresponds to the q multiplier increasing. This is why the slow increase is longer as n increases.

Another interesting point is that when $n \geq q$, $\delta = 1$ and the space required by the structure is greater than the simple bit-vector strategy. For this reason, the values obtained by this method are only interesting where $n < q$. We will see this fact again in the comparative analyses.

3 Easy Lower Bound: If there is a $(U, n; s, q)$-BPDS then $s \geq c_0 U^{1/q}$

We now present our first lower bound. This argument was also devised as an introduction to BPDS lower bounds by William Gasarch. This is the first time this argument has appeared in literature and we present both the algorithm and a detailed analysis of its constant factors for comparison with other algorithms.

The key argument for this lower bound is this: Say two sets, $A_1 = \{15, 64\}$ and $A_2 = \{15\}$, have the same representation using the same bits (e.g., $CELL[5] = 1$ and $CELL[44] = 1$). Then the query "$64 \in A$?" cannot be answered and therefore, no BPDS would store A_1 and A_2 this way. The lower bound cleverly chooses BPDS parameters that will force this condition to reach a contradiction and argue that no such BPDS can exist.

3.1 The Lower Bound

Theorem 3.1 *Let $n, q \in \mathbb{N}$. Let M be the least number such that*

$$\binom{M}{0} + \cdots + \binom{M}{n} \geq 2^q + 1$$

If there is an $(U, n; s, q)$-BPDS then $\binom{s}{q} \geq \frac{U}{M}$.

Proof:

Assume, to reach a contradiction, that there is a $(U, n; s, q)$-BPDS where $\binom{s}{q} < \frac{U}{M}$

Let $f : [U] \rightarrow \binom{[s]}{q}$ be a function that maps u to the set of queries the membership algorithm asks. Since $\binom{s}{q} < \frac{U}{M}$ there must be some M elements of $[U]$ that map to the same set of queries. Let those M elements of $[U]$ be $W = \{u_1, \ldots, u_M\}$ and let the set of bit queries be $X = \{x_1, \ldots, x_q\}$.

Next, map every potentially stored subset, A, of W to the way the bits of X are set to store A. The domain of this function is

$$\binom{[W]}{0} \cup \cdots \cup \binom{[W]}{n}$$

This domain has the size

$$\binom{M}{0} + \cdots + \binom{M}{n} \geq 2^q + 1$$

The codomain has size 2^q (ways to set q bits). Hence there must be two sets that map to the exact same bit pattern. This causes a contradiction because the BPDS cannot distinguish between those two sets.

Therefore, for any $(U, n; s, q)$-BPDS, $\binom{s}{q} \geq \frac{U}{M}$.

▮

A simple corollary now gives the lower bound.

Corollary 3.2 *Let $n, q \in \mathbb{N}$, There exists U_0 and c_0 such that, for all $U \geq U_0$, if there is an $(U, n; s, q)$-BPDS then $s \geq cU^{1/q}$.*

Proof:

Let M be the least number such that

$$\binom{M}{0} + \cdots + \binom{M}{n} \geq 2^q + 1.$$

Note that M is a constant. By Theorem 3.1 $\binom{s}{q} \geq \frac{U}{M}$. Note that $\binom{s}{q} \leq s^q$. Hence we have

$$s^q \geq \binom{s}{q} \geq \frac{U}{M}.$$

$$s \geq \left(\frac{1}{M}\right)^{1/q} U^{1/q}$$

Let $c_0 = (\frac{1}{M})^{1/q}$.

∎

3.2 Actual Values

To calculate the actual lower bounds on s, depending on U, q, and n, we first have to calculate values of the constant M, which depends only on q and n. Table 3 shows values of M calculated for different q and n.

Using s^q for $\binom{s}{q}$ as in the last section is not a very good approximation. Before calculating a table of s-values, we give the following improvement:

$$\binom{s}{q} = \frac{s!}{q!(s-q)!} = \frac{s(s-1)(s-2)\cdots(s-q+1)}{q!} \leq \frac{s^q}{q!}$$

$n \setminus q$	3	4	5	6	7	8	9	10	11	12	13	14	15
1	8	16	32	64	128	256	512	1024	2048	2^{12}	2^{13}	2^{14}	2^{15}
2	4	6	8	11	16	23	32	45	64	91	128	181	256
3	4	5	6	8	9	12	15	19	24	30	37	47	59
4	4	5	6	7	8	10	11	13	16	19	22	26	31
5	4	5	6	7	8	9	10	12	13	15	17	19	22
6	4	5	6	7	8	9	10	11	12	14	15	17	19
7	4	5	6	7	8	9	10	11	12	13	14	16	17
8	4	5	6	7	8	9	10	11	12	13	14	15	16
9	4	5	6	7	8	9	10	11	12	13	14	15	16
...	4	5	6	7	8	9	10	11	12	13	14	15	16

Table 3: Values of M for different q and n.

Using this instead yields a much closer approximation:

$$\frac{s^q}{q!} \geq \frac{U}{M} \Rightarrow s \geq \left(\frac{q! \cdot U}{M}\right)^{1/q}$$

Again using $U = 2^{32} - 1$, we can calculate concrete lower bounds for s. These lower bounds are given in Table 4. Unfortunately, these bounds are not very interesting because they are still so low. We will revisit these values for comparison in Section 8, which gives an alternative generalized lower bound.

$n \setminus q$	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1477	284	111	61	41	31	25	21	19	17	16	15	15
2	1861	363	146	81	55	41	34	29	26	24	22	21	20
3	1861	379	154	86	59	45	37	32	28	26	24	23	22
4	1861	379	154	88	60	46	38	33	29	27	25	24	23
5	1861	379	154	88	60	46	38	33	30	27	26	24	23
6	1861	379	154	88	60	46	38	33	30	27	26	25	24
7	1861	379	154	88	60	46	38	33	30	28	26	25	24
8	1861	379	154	88	60	46	38	33	30	28	26	25	24
9	1861	379	154	88	60	46	38	33	30	28	26	25	24
...	1861	379	154	88	60	46	38	33	30	28	26	25	24

Table 4: Lower bounds on s, for different q and n using $U = 2^{32} - 1$.

4 Easy Probabilistic BPDS: An Adaptation of the Easy Upper Bound

In this section, we give an introduction to probabilistic bit probe data structures. Probabilistic data structures try to improve upper bounds, either by reducing the number of queries or by reducing the amount of space required, by taking advantage of some reasonable tolerance for failures (incorrect answers). We allow the query function to "flip coins", to improve the number of queries or space it takes to answer the question "$x \in A$"?

This is the only section in that deals with probabilistic methods and is included as an important variation of our main topic. This adaptation of the EASY upper bound is new and was produced by Ryan Blue. We will now define the probabilistic bit probe model, still in the non-adaptive case.

Def 4.1 A one-sided *Probabilistic BPDS*, or $(U, n, \epsilon; s, q)$-Prob-BPDS, is a BPDS where the membership algorithm, MEM, is given a random string and decides membership with a probability of being wrong less than some ϵ. That is,

- $\Pr(MEM(u,r) = IN | u \notin A) < \epsilon$
- $\Pr(MEM(u,r) = OUT | u \in A) = 0$

In a Prob-BPDS, MEM consists of:

1. A function that takes as input $u \in [U]$ and $r \in \{0,1\}^m$ and outputs $(i_1, \ldots, i_{q'})$.
2. A function, MEM, that takes as input u and r and outputs a boolean function $f_{u,r}$ on q' boolean variables and concludes

$$u \in U \text{ iff } f_{u,r}(CELL[i_1], \ldots, CELL[i_{q'}]) = IN$$

   This MEM process decides with probabilities:

$$Pr(MEM(u,r) = IN | u \notin A) < \epsilon$$

$$Pr(MEM(u,r) = OUT | u \in A) = 0$$

Note 4.2 We use q' above because we think of $q' < q$, where q is the number of queries in the deterministic "version" of MEM. This allows us to define the probabilistic version in terms of the deterministic version.

Consider the following probabilistic adaptation of the easy upper bound in Section 2:

Theorem 4.3 *Let $n, q \in \mathbb{N}$. Let $\delta = \frac{1}{\lceil q/n \rceil}$. There exists constants U_0, c_0 such that, for all $U \geq U_0$, there is a $(U, n, \epsilon; c_0 U^{\delta}, q-1)$-Prob-BPDS, where $\epsilon > 1/q$*

Proof:

Setting up the Data Structure:

- Use the same procedure as the deterministic BPDS.

Making a Query:

- Select, at random, $q' = q - 1$ bits of B_u.
- If all q' bits are 1, then output IN, else OUT.

Why does this work?

- The probability that this algorithm concludes incorrectly, deciding that $u \in A$ when it is not, is the probability that each $CELL[i]$ queried collided with another value stored in the data structure. Because we know that there is at least one cell that never collides with other values, this cell was not chosen.
- The probability of NOT choosing a particular cell is $1/q$. So:

$$Pr(MEM(u, r) = IN | u \notin A) \leq 1/q < \epsilon$$

- The probability that this algorithm concludes incorrectly, deciding that $u \notin A$ when it is, equals 0. We know that if this algorithm decides OUT, then there was at least one $CELL[i]$ that was 0, and u would be OUT in the original data structure, regardless of the unchecked cell.

∎

This alteration of EASY is able to reduce the number of queries by one. This technique, however, uses the same amount of space as the original algorithm[2] because any of the bit positions could be chosen and queried. Other Probabilistic-BPDS can have much better results than this demonstration, reducing both storage space and number of queries, but such structures are not discussed in this paper. See [6] for more extensive results on probabilistic methods.

[2]See Section 3 for a space analysis and table of values.

5 Upper Bound: There is a $(U, n; s, 4)$-BPDS where $s = O(n^{1/6}U^{5/6})$

Now we move on to upper bounds from literature. This section presents the first of our upper bounds. This upper bound is not constructive (or *explicit*) like the EASY upper bound, but instead relies on probabilistic techniques to argue that a data structure using space $\leq c_0 \cdot n^{1/6}U^{5/6}$ must exist. This section is a stepping stone to the next section, which uses the same logic but more complicated means to achieve a better upper bound.

Our contribution in this section is the thorough proof write-up and the space comparison values given at the end of the section.

5.1 Combinatorial Set Up

We first need to define a graph that we will use frequently for the rest of the paper.

This proof introduces a bipartite graph[3] associated with a bit probe data structure[4] that is key to many proofs. Figure 1 shows an example of such a graph for a 3-query BPDS.

Notation 5.1 A bipartite graph of the form $([U], [s], E)$ will be denoted (U, s, E).

Def 5.2 A bipartite graph (U, s, E) where every element of U has degree q is called *q-regular*.

[3] A bipartite graph is a graph that can be divided into two disjoint sets of vertices such that every edge connects a vertex in one set to some vertex in the other

[4] [6] uses hypergraphs in places, however, we use only the equivalent bipartite graphs.

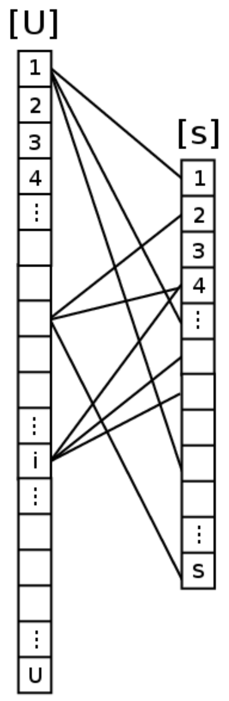

Figure 1: An example (U, s, E) graph associated with a 3-query bit probe data structure.

Def 5.3 Assume we have a $(U, n; s, q)$-BPDS. We associate to it the following bipartite graph, $G = (U, s, E)$, where

$$E = \{(u, x) : \text{ to answer “}u \in A\text{?” one of the bit-queries is } x. \}$$

Every $u \in [U]$ has degree q (this graph is q-regular). Note that not every node in $[s]$ necessarily has degree q and that the bipartite graph stores only the bit locations that are used as input to f_u. It has no information about the function f_u itself.

We will refer to this graph as *the graph associated with the bit-probe data structure* or as *the BPDS's corresponding graph.*

Def 5.4 Let $G = (U, s, E)$ be a bipartite graph. Let $A \subseteq U$. Then G_A is the labeled bipartite graph where the elements of A are labeled IN and the elements in $[U] - A$ are labeled OUT.

Def 5.5 Let $a, b, q \in \mathbb{N}$ such that $0 < a, b < q$. Let $G = (U, s, E)$ be a q-regular labeled bipartite graph. Let $A \subseteq U$. An *(a, b)-coloring of G_A* is a partial 2-coloring of $[s]$, using the colors $\{0, 1\}$ such that the following occurs:

1. Every $u \in A$ then at least a neighbors of u are colored 1.
2. Every $u \notin A$ then at least b neighbors of u are colored 0.

We allow the 2-coloring of $[s]$ to be partial because all we care about is the conclusion that IF $u \in A$ then ... and IF $u \notin A$ then ... and a partial coloring is sufficient to draw such conclusions.

Def 5.6 Let $a, b, q \in \mathbb{N}$ such that $0 < a, b < q$. Let $G = (U, s, E)$.

1. G is (a, b, q)-useful if it is q-regular and has the following property: for any subset $A \subseteq [U]$, G_A is (a, b)-colorable.

2. G is $(n; a, b, q)$-useful if it is q-regular and has the following property: for any subset $A \subseteq [U]$, $|A| \leq n$, G_A is (a, b)-colorable.

(2) is defined here because the constraints on (1) would be too strong for this proof to work. Because we are only storing n-sized subsets of $[U]$, (2) is sufficient and will work later on.

Lemma 5.7 *If there exists a 4-regular* $(n; 3, 2, 4)$*-useful bipartite graph* $G = (U, s, E)$ *then there is a* $(U, n; s, 4)$*-BPDS.*

Proof:

Setup:

- Let $A \subseteq U$, $|A| \leq n$. Let COL be the $(3, 2)$-coloring of G_A. For each $x \in [s]$ that is colored 0 (1) let the corresponding bit be 0 (1).

Query:

- To determine if $u \in A$ ask the 4 bit-queries that correspond to the elements in $N(u)$. If at least 3 of them are 1 then answer IN. If at most 2 of them are 1 then answer OUT.

Why does this work?

- If $u \in A$ then A was labeled IN. Hence at least 3 of its neighbors are colored 1. Hence the algorithm will return IN.
- If $u \notin A$ then A was labeled OUT. Hence at least 2 of its neighbors are colored 0. Hence at most 2 neighbors are colored 1. Hence the algorithm will return OUT.

▮

Lemma 5.7 can be generalized for any value of q, so our job is reduced to finding such a bipartite graph. We approach this problem by introducing and using an *expansion* property of bipartite graphs next:

5.2 $\frac{14^+}{5}$-Expansion Implies $(3,2,4)$-Useful

Def 5.8 Let $\alpha > 1$. Let $G = (U, s, E)$ be a bipartite graph.

1. G has *expansion* α if for every $A \subseteq [U]$, $|N(A)| \geq \alpha|A|$.
2. G has *expansion* α^+ if for every $A \subseteq [U]$, $|N(A)| > \alpha|A|$.

Looking at the definition of expansion it looks like you would need $s \geq U$ to have an α-expanding graph. This is true, and, therefore, it would seem that such graphs are not useful to us. But they will be used later as a subroutine of what we need to do (and as a subroutine in Section 6 as well).

This is why we defined $(n; 3, 2, 4)$-useful earlier in addition to $(2, 3, 4)$-useful. The latter entails α-expanding graphs with $s \geq U$, while the former enables us to get $s < U$.

Lemma 5.9 *Let $G = (U, s, E)$ be a 4-regular bipartite graph with expansion $\frac{14^+}{5}$.*

1. *G is $(3, 2, 4)$-useful.*
2. *Let $A \subseteq [U]$. Informally, we want to say that a partial $(3, 2)$-coloring of G_A can be extended to a full $(3, 2)$-coloring of G_A. We proceed formally. Let $A \subseteq U$. Assume the right hand side of G has been partially $(3, 2)$-colored such that*

   (a) if $u \in U$ is IN then either at least 3 elements of $N(u)$ are colored 1 or no elements of $N(u)$ are colored 0, and

   (b) if $u \in U$ is OUT then either at least 2 elements of $N(u)$ are colored 0 or no elements of $N(u)$ are colored 1.

   Then this partial coloring can be extended to a full $(3, 2)$ coloring.

Proof:

We prove part (2). Part (1) follows. Let $A \subseteq [U]$. Assume there already is a partial coloring as described in the premise. The following greedy algorithm, henceforth GREEDY, will complete the $(3, 2)$-coloring:

1. Initialize $ACTIVE$ to be the union of the following two sets.

$$\{u \in U : \ u \text{ is labeled IN and} \leq 2 \text{ of its neighbors are colored } 1 \ \}$$

$$\{u \in U : \ u \text{ is labeled OUT and} \leq 1 \text{ of its neighbors are colored } 0 \ \}$$

```
2. Initialize $NCOL$ to be

   $$NCOL = \{x \in [s] : \text{ } x \text{ has not been colored yet } \}.$$

3. For all $x \in NCOL$
   (a) If $N(x) \cap ACTIVE$ are all IN then $COL(x) = 1$ and $NCOL = NCOL - \{x\}$.
   (b) If $N(x) \cap ACTIVE$ are all OUT then $COL(x) = 0$ and $NCOL = NCOL - \{x\}$.
4. For all $u \in ACTIVE$
   (a) If at least 3 elements of $N(u)$ are colored 1 then $ACTIVE = ACTIVE - \{u\}$.
   (b) If at least 2 elements of $N(u)$ are colored 0 then $ACTIVE = ACTIVE - \{u\}$.
5. If $ACTIVE \neq \emptyset$ then GOTO Step 3
```

Clearly all $u \notin ACTIVE$ that are labeled IN (OUT) have at least 3 (2) neighbors colored 1 (0). We need to show that eventually $ACTIVE = \emptyset$. (Note that this is all we need- we do not need that $NCOL = \emptyset$ since the coloring can be partial.) We show that so long as $ACTIVE \neq \emptyset$ either some $u \in [U]$ will become inactive or some $x \in [s]$ will get colored.

Assume $ACTIVE \neq \emptyset$ but during an iteration of the algorithm no element of $[s]$ is colored and no element of $[U]$ is made inactive. We will get a contradiction. Let $ACTIVE = ACT_{IN} \cup ACT_{OUT}$ where the IN (OUT) elements of

$ACTIVE$ are ACT_{IN} (ACT_{OUT}).

We count $|N(ACT_{IN} \cup ACT_{OUT})|$ in two different ways.

1. Every elements of ACT_{IN} has 4 neighbors. Look at $u \in ACT_{OUT}$. We look at the neighbors of u and determine how many of them are not already counted in $N(ACT_{IN})$. No neighbor of u is colored 1 since $u \in ACT_{OUT}$. At most one neighbor of u is colored 0 (if two were colored 0 then $u \notin ACTIVE$). Of the non-colored neighbors of u all of them are in $N(ACT_{IN})$, else they would have been colored. Hence

   $$|N(ACT_{IN} \cup ACT_{OUT})| \le 4|ACT_{IN}| + |ACT_{OUT}|.$$

   Hence (we'll need this later)

   $$2|N(ACT_{IN} \cup ACT_{OUT})| \le 8|ACT_{IN}| + 2|ACT_{OUT}|.$$

2. Every elements of ACT_{OUT} has at most 4 neighbors. Look at $u \in ACT_{IN}$. We look at the neighbors of u and determine how many of them are not already counted in $N(ACT_{OUT})$. No neighbor of u is colored 0 since $u \in ACT_{IN}$. At most two neighbor of u are colored 1 (if three were colored 1 then $u \notin ACTIVE$). Of the non-colored neighbors of u all of them are in $N(ACT_{OUT})$, else they would have been colored. Hence

   $$|N(ACT_{IN} \cup ACT_{OUT})| \le 2|ACT_{IN}| + 4|ACT_{OUT}|.$$

   Hence (we'll need this later)

$$3|N(ACT_{IN} \cup ACT_{OUT})| \le 6|ACT_{IN}| + 12|ACT_{OUT}|.$$

Adding together the two equations that we claimed we would need later you get

$$5|N(ACT_{IN} \cup ACT_{OUT})| \le 14|ACT_{IN}| + 14|ACT_{OUT}|.$$

or

$$|N(ACT_{IN} \cup ACT_{OUT})| \le \frac{14}{5}(|ACT_{IN}| + |ACT_{OUT}|).$$

Hence if $X = ACT_{IN} \cup ACT_{OUT}$ then $|N(X)| \le \frac{14}{5}|X|$. This contradicts G being $\frac{14^+}{5}$-expanding. ▮

This looks good: if we can find a graph $G = (U, s, E)$ that is 4-regular and $\frac{14^+}{5}$-expanding then it will be $(3, 2, 4)$-useful. However, since the expansion condition as we have stated it applies even to the entire set U, the resulting data structure would use space greater than $\frac{14U}{5}$, which is no good. But we only need an $(n; 3, 2, 4)$-useful graph, and for that, a weaker condition will suffice.

5.3 $(2n, \frac{14^+}{5})$-Expansion implies $(n; 3, 2, 4)$-Useful

Def 5.10 Let $\alpha > 1$ and $L \in \mathbb{N}$.

1. $G = (U, s, E)$ has *expansion* (L, α) if for every $A \subseteq [U]$, $|A| \le L$, $|N(A)| \ge \alpha|A|$.

2. $G = (U, s, E)$ has *expansion* (L, α^+) if for every $A \subseteq [U]$, $|A| \le L$, $|N(A)| > \alpha|A|$.

Lemma 5.11 *Let $G = (U, s, E)$ be a 4-regular bipartite graph with expansion $(2n, \frac{14^+}{5})$. Then G is $(n; 3, 2, 4)$-useful.*

Proof:

Let $A \subseteq [U]$ such that $|A| \le n$. We exhibit a $(3, 2)$ coloring of G_A. Actually we just do the algorithm from Lemma 5.9. We need to show that eventually $ACTIVE = \emptyset$.

Assume $ACTIVE \ne \emptyset$ but during an interaction of the algorithm no element of $[s]$ is colored and no element of $[U]$ is made inactive. We will get a contradiction. Let $ACTIVE = ACT_{IN} \cup ACT_{OUT}$ where the IN (OUT) elements of $ACTIVE$ are ACT_{IN} (ACT_{OUT}).

Claim 1 $|ACT_{OUT}| < |ACT_{IN}|$.

Proof:

Assume, by way of contradiction, that $|ACT_{OUT}| \ge |ACT_{IN}|$.

Choose $V \subseteq ACT_{OUT}$ of exactly the same size as ACT_{IN}. Note that $|V| = |ACT_{IN}| \le n$ (since originally $|A| \le n$), so

$$|ACT_{IN} \cup V| \le 2n$$

Hence, since G is $(2n, \frac{14^+}{5})$-expanding,

$$|N(ACT_{IN} \cup V)| > \frac{14}{5}(|ACT_{IN}| + |V|)$$

Since $|V| = |ACT_{IN}|$ we have

$$|N(ACT_{IN} \cup V)| > \frac{28}{5}(|ACT_{IN}|)$$

This is what we will contradict.

By the same reasoning used in Lemma 5.9 we have that

$$|N(ACT_{IN} \cup V)| \leq 4|ACT_{IN}| + |V| = 5|ACT_{IN}|$$

Note that we have

$$\frac{28}{5}|ACT_{IN}| < |N(ACT_{IN} \cup V)| \leq 5|ACT_{IN}|$$

This is a contradiction.

▮

The set of active elements of $[U]$ has size $\leq 2n$. Hence the bipartite graph that is left to color has $\leq 2n$ elements in the left hand side. By the premise this graph is α-expanding. It is partially colored. By Lemma 5.9 it can be extended to a full coloring. Formally this contradicts the assumption that the coloring stopped. ▮

Note 5.12 Lemma 5.11 used the $\alpha > \frac{14}{5}$. This is not optimal.

5.4 The Upper Bound: There exists a Bipartite Graph Such That...

In the last sections, we showed that if there is a $(2n; 14/5^{+})$-expanding graph, then that graph is $(n; 3, 2, 4)$-useful. Now we need to show that such a graph exists to complete the upper bound proof.

Lemma 5.13 *There exists constants c, U_0 such that the following is true. For every $\delta > 0$, for every U, n with $U \geq n$, and $U \geq U_0$, for every $s \geq cn^{\delta/(1+\delta)}U^{1/(1+\delta)}$ there is a 4-regular $(2n, (3-\delta)^{+})$-expanding bipartite graph $G = (U, s, E)$.*

Proof:

We show the graph exists by the probabilistic method. To do this, we take appropriately sized bipartite graphs with vertices in $[U]$ and $[s]$, and generate random edges such that the graph is 4-regular. Then, we probabilistically show that at least one randomly chosen graph will have the desired expansion, $3 - \delta$.

Let $\Delta = (3 - \delta)$ to simplify formulas. Also note that

$$\left(\frac{m}{k}\right)^{k} \leq \binom{m}{k} \leq \left(\frac{e \cdot m}{k}\right)^{k} \tag{1}$$

(The e really is the e you know: $2.718\ldots$)

For every $u \in [U]$ pick 4 elements of $[s]$ at random to be its neighbors in the bipartite graph. Consider subsets of $[U]$ that are size $2n$ and smaller.

We bound the expected number of X such that $|X| \leq 2n$ and $|N(X)| \leq \Delta|X|$. Since $|N(X)|$ is an integer this is equivalent to $|N(X)| \leq \lfloor \Delta|X| \rfloor$. Let $1 \leq r \leq 2n$.

We first bound the expected number of X where $|X| = r$, such that $|N(X)| < \lfloor \Delta r \rfloor$.

There are $\binom{s}{\lfloor \Delta r \rfloor}$ subsets of $[s]$ of size $\lfloor \Delta r \rfloor$. Let Y be one of them. What is the probability that $N(X) \subseteq Y$? It is

$$\left(\frac{\binom{\lfloor \Delta r \rfloor}{4}}{\binom{s}{4}} \right)^r$$

Using equation 1, we can bound this probability

$$\left(\frac{\binom{\lfloor \Delta r \rfloor}{4}}{\binom{s}{4}} \right)^r \leq \left(\frac{\left(\frac{e\lfloor \Delta r \rfloor}{4}\right)^4}{\left(\frac{s}{4}\right)^4} \right)^r = \left(\frac{e \lfloor \Delta r \rfloor}{s} \right)^{4r}$$

Hence the expected number of such X (of size r) is bounded above by

$$\binom{s}{\lfloor \Delta r \rfloor} \left(\frac{e \lfloor \Delta r \rfloor}{s} \right)^{4r}$$

Hence the total number of such X (of size $1 \leq r \leq 2n$) is

$$\sum_{r=1}^{2n} \binom{U}{r} \binom{s}{\lfloor \Delta r \rfloor} \left(\frac{e \lfloor \Delta r \rfloor}{s} \right)^{4r}.$$

Again, by equation 1:

$$\binom{U}{r} \leq \left(\frac{eU}{r} \right)^r$$

$$\binom{s}{\lfloor \Delta r \rfloor} \leq \left(\frac{es}{\lfloor \Delta r \rfloor}\right)^{\lfloor \Delta r \rfloor}$$

Putting this all together, we get that the expected number of X is bounded above by

$$\sum_{r=1}^{2n} \left(\frac{eU}{r} \left(\frac{es}{\lfloor \Delta r \rfloor} \right)^{\Delta} \left(\frac{e \lfloor \Delta r \rfloor}{s} \right)^{4} \right)^{r}$$

We look at the summand.

$$\left(\frac{eU}{r} \left(\frac{es}{\lfloor \Delta r \rfloor} \right)^{\Delta} \left(\frac{e \lfloor \Delta r \rfloor}{s} \right)^{4} \right)^{r} \leq \left(\frac{e^{\Delta+5}U}{r} \frac{(\lfloor \Delta r \rfloor)^{4-\Delta}}{s^{4-\Delta}} \right)^{r} \leq \left(\frac{e^{\Delta+5}\Delta^{4-\Delta}Ur^{3-\Delta}}{s^{4-\Delta}} \right)^{r}.$$

Since $\Delta = 3 - \delta$ and $r \leq 2n$ we get that this quantity is bounded above by

$$\left(\frac{e^{8-\delta}(3-\delta)^{1+\delta}Ur^{\delta}}{s^{1+\delta}} \right)^{r} \leq \left(\frac{e^{8-\delta}(3-\delta)^{1+\delta}U(2n)^{\delta}}{s^{1+\delta}} \right)^{r} = \left(\frac{e^{8-\delta}(3-\delta)^{1+\delta}2^{\delta}U(n)^{\delta}}{s^{1+\delta}} \right)^{r}$$

Let c be a bound on $e^{8-\delta}(3-\delta)^{1+\delta}2^{\delta}$ that is independent of n, U and even δ. Then the quantity above is bounded above by

$$\left(\frac{cUn^{\delta}}{s^{1+\delta}} \right)^{r}$$

Hence we have that the expected number of such X is bounded above by

$$\sum_{r=1}^{2n} \left(\frac{cUn^{\delta}}{s^{1+\delta}} \right)^{r}.$$

The sum is geometric. If the summand is less than $\frac{1}{2}$, then the summation must be less than 1, which is what we're after. This yields

$$\begin{aligned} \tfrac{cUn^\delta}{s^{1+\delta}} &\leq \tfrac{1}{2} \\ 2cUn^\delta &\leq s^{1+\delta} \\ (2cUn^\delta)^{1/(1+\delta)} &\leq s \end{aligned}$$

Let $c_0 = 2c$. We have that if $s \geq c_0 n^{\delta/(1+\delta)} U^{1/(1+\delta)}$ there is a 4-regular $(2n, 3-\delta)$-expanding bipartite graph. ▮

Theorem 5.14 *There exists U_0 and c_0 such that, for all $U \geq U_0$, for all n, there is a $(U, n; s, 4)$-BPDS where $s = \lceil c_0 n^{1/6} U^{5/6} \rceil$.*

Proof: This follows from Lemmas 5.7, 5.11, and 5.13 with $\delta = 1/5$.

▮

5.5 Actual Values

The actual constant term can be obtained from Lemma 5.13:

$$c_0 = 2c = (6 - 2\delta)^{1+\delta} \cdot e^{8-\delta}$$

and

$$s \geq c_0 \cdot n^{\frac{\delta}{1+\delta}} \cdot U^{\frac{1}{1+\delta}}$$

To get a more precise answer, we use c that depends on δ. When $\delta = 1/5$

n	EASY	$n^{1/6}U^{5/6}$
2	$2.634 \cdot 10^5$	$2.306 \cdot 10^{12}$
3	$2.634 \cdot 10^5$	$2.467 \cdot 10^{12}$
4	$1.718 \cdot 10^{10}$	$2.589 \cdot 10^{12}$
5	$1.718 \cdot 10^{10}$	$2.687 \cdot 10^{12}$
6	$1.718 \cdot 10^{10}$	$2.770 \cdot 10^{12}$
7	$1.718 \cdot 10^{10}$	$2.842 \cdot 10^{12}$
8	$1.718 \cdot 10^{10}$	$2.906 \cdot 10^{12}$
9	$1.718 \cdot 10^{10}$	$2.963 \cdot 10^{12}$
10	$1.718 \cdot 10^{10}$	$3.016 \cdot 10^{12}$

Table 5: Values of s using different n for both EASY and Expansion-14/5$^+$ algorithms. ($U = 2^{32} - 1$, $q = 4$)

(expansion $3 - 1/5 = 14/5$).

$$c_0 = \left(\frac{28}{5}\right)^{6/5} \cdot e^{39/5}$$

Using this constant value, Table 5 compares values of s for different n between the EASY algorithm and the Expansion algorithm. For low n values, EASY has much better space values. We also include Table 6, which shows s with powers of 2 as n. The constant factor—and thus space required—for EASY is better for higher values as well.

In the next section, we will get a better upper bound by using a smaller expansion. We will then revisit these comparisons.

n	EASY	$n^{1/6}U^{5/6}$
8	$1.718 \cdot 10^{10}$	$2.906 \cdot 10^{12}$
16	$1.718 \cdot 10^{10}$	$3.261 \cdot 10^{12}$
32	$1.718 \cdot 10^{10}$	$3.661 \cdot 10^{12}$
64	$1.718 \cdot 10^{10}$	$4.109 \cdot 10^{12}$
128	$1.718 \cdot 10^{10}$	$4.613 \cdot 10^{12}$
256	$1.718 \cdot 10^{10}$	$5.178 \cdot 10^{12}$
512	$1.718 \cdot 10^{10}$	$5.812 \cdot 10^{12}$
1024	$1.718 \cdot 10^{10}$	$6.523 \cdot 10^{12}$
2048	$1.718 \cdot 10^{10}$	$7.322 \cdot 10^{12}$
4096	$1.718 \cdot 10^{10}$	$8.219 \cdot 10^{12}$
8192	$1.718 \cdot 10^{10}$	$9.226 \cdot 10^{12}$

Table 6: Values of s using different powers of 2 as n for both EASY and Expansion-$14/5^+$ algorithms. ($U = 2^{32} - 1$, $q = 4$)

6 Upper Bound: There is a $(U, n; s, 4)$-BPDS where $s = O(n^{1/4}U^{3/4})$

In the previous section, we used the $\frac{14^+}{5}$-expansion to guarantee that the (greedy) algorithm would not stop until the ACTIVE set was exhausted. To get a smaller expansion, and thus smaller s required to store A, we need to show what happens when the algorithm stops but the $ACTIVE$ set is NOT empty.

The following section shows that a $\frac{8^+}{3}$-expanding graph is $(3, 2)$-colorable.

Note 6.1 [1] also shoes that 8/3 is the minimal expansion to guarantee a $(3, 2)$-coloring by constructing a graph that is one vertex short of an 8/3 expansion and cannot be $(3, 2)$-colored.

This is based on Theorem 4.3 in [1].

6.1 A $\frac{8^+}{3}$-expanding graph is $(3,2)$-colorable

Def 6.2 Let $G = (U, s, E)$ be a bipartite graph. Let *col* be a partial (p, q)-coloring of G. Let $N_i(u)$ ($u \in [U]$) denote the number of neighbors ($\in [s]$) colored i. The *demand* of a vertex $u \in [U]$ is:

1. if u is IN, $demand(u) = p - N_1(u)$
2. if u is OUT, $demand(u) = q - N_0(u)$

Def 6.3 Let $G = (U, s, E)$ be a bipartite graph. Let $W \subseteq [U]$. A *preliminary coloring* of W is a partial (p, q)-coloring where:

1. For all $i \in [s]$, $col(i) = 1$ if all of its neighbors in W are IN
2. For all $i \in [s]$, $col(i) = 0$ if all of its neighbors in W are OUT

The set W is *demanding* if $\forall u \in W$, $demand(u) > 0$.

Note 6.4 This definition of *demanding* describes the situation that exists after the greedy algorithm from Section 5.2 stops and any vertices in $NCOL$ that can be colored are colored. $ACTIVE$ forms an demanding set. The definition, however, is defined independently (from the algorithm), so we can use it in Lemma 6.6.

Def 6.5 A graph $G = (U, s, E)$ satisfies the *counting condition* if, following the preliminary coloring, for every choice of demanding W, the number of remaining (uncolored) vertices in $[s]$ is at least as large as the sum of the demands of W. For all $W \subseteq [U]$:

$$|N(W) - N_1(W) - N_0(W)| \geq \sum_{u \in W} demand(u)$$

Lemma 6.6 *Any bipartite graph, $G = (U, s, E)$, with expansion $\alpha \geq 8/3$ satisfies the counting condition.*

Proof: We proceed with proof by contradiction. Let G be as in the statement of the lemma and let W be any demanding set. Pre-color G using W.

Let $C_i, i \in \{0, 1, 2\}$ denote the number of IN vertices with i neighbors that are pre-colored. Similarly, let $B_j, j \in \{0, 1\}$ denote the number of OUT vertices with i neighbors that are pre-colored.

We will use m to denote the number of neighbors of W that are not pre-colored ($m = |N(W) - N_1(W) - N_0(W)|$).

Assume, to reach contradiction, that the set W violates the counting condition:

$$m < \sum_{u \in W} demand(u)$$

Next, we rewrite this equation in terms of the C_i and B_j by noting that any two $u \in W$ that contribute to the same C_i (B_j) have the same demand: $3 - i$ $(2 - j)$

$$m < 3C_0 + 2C_1 + 1C_2 + 2B_0 + 1B_1 \tag{2}$$

The second key equation comes from bounding m_2 by using a counting technique similar to counting ACT_{IN}. Every vertex counted in m_2 must be connected to an IN and an OUT node in $[U]$. For every OUT node with no pre-colored neighbors, there are at most 4 distinct vertices in $[s]$ and for every OUT node with 1 pre-colored neighbor, there are at most 3 distinct neighbors. Therefore,

$$m \leq 4B_0 + 3B_1 \tag{3}$$

Adding two of equation (1) with one of equation (2) yields:

$$3m < 6C_0 + 4C_1 + 2C_2 + 8B_0 + 5B_1$$

or, when reduced,

$$m < 2C_0 + \frac{4}{3}C_1 + \frac{2}{3}C_2 + \frac{8}{3}B_0 + \frac{5}{3}B_1 \tag{4}$$

The total number of vertices, $N(W)$, in terms of m, C_i, and B_j for counting is

$$N(W) \leq C_1 + 2C_2 + B_1 + m$$

Substituting (3) for m,

$$N(W) < 2C_0 + \frac{7}{3}C_1 + \frac{8}{3}C_2 + \frac{8}{3}B_0 + \frac{8}{3}B_1 < \frac{8}{3}|W| \tag{5}$$

Equation (4) contradicts the $\frac{8}{3}$-expansion property; therefore, G satisfies the counting condition.

∎

Now we are almost ready to present the final algorithm to produce a $(3, 2)$-coloring. We use the GREEDY algorithm as a sub-routine, and we also use a second sub-routine, MATCH, which finds an optimal matching on a bipartite graph by using alternating paths.

Def 6.7 An *alternating path* is a list of edges in a graph, $(x_1, y_1), (x_2, y_2), \ldots, (x_l, y_l)$, where (x_i, y_i) and (x_i, y_{i+1}) are edges in the graph for all $1 \leq i < l$. In other words, an alternating path is a set of edges formed by taking every other edge of a path in G.

Alternating paths can be used to find an optimal matching[5]. This is the approach used by MATCH:

1. Input $G, MATCHABLE, COL$.
2. Let $G' = (MATCHABLE, N(MATCHABLE), E')$ where

   $$E' = \{(x, y) \in E : x \in MATCHABLE \wedge y \in N(MATCHABLE)\}$$

3. Let $M = (x, y) \in E'$, where (x, y) is any edge. M will be an alternating path in E', with edges $(x_1, y_1), \ldots, (x_l, y_l)$.
4. Let $UNMATCHED = \{x \in MATCHABLE : \nexists (x, y) \in M$.
5. If $UNMATCHED = \emptyset$, GOTO Step 7.

[5]For further reading on Bipartite Matchings, see [7]

6. For $x \in UNMATCHED$ do

    - For $y \in \{y \in N(MATCHABLE) : \nexists (x,y) \in M\}$ do

        – If $(x, y_1) \in E'$ and $(x_l, y) \in E'$ (x and y can be used to form a larger alternating path), then let

$$M = \{(x, y_1), (x_1, y_2), \ldots, (x_{l-1}, y_l), (x_l, y)\}$$

        – GOTO Step 4.

7. For each $(x, y) \in M$, let $COL(y) = 1$ if x is labeled IN and $COL(y) = 0$ if x is labeled OUT.

Note 6.8 MATCH may not terminate on any input, however, in the final algorithm, MATCH is always run with inputs for which Hall's Theorem guarantees there is a perfect matching.

The final algorithm is:

1. Run GREEDY on $G, ACTIVE, COL$.
2. If $|ACTIVE| = 0$, output COL.
3. Find the largest subset, X, of $ACTIVE$ such that $\sum_{u \in X} demand(u) > N(X)$.
4. Let $MATCHABLE = ACTIVE - X$.
5. Run MATCH on $G, MATCHABLE, COL$.
6. Let $ACTIVE = X$.

7. If $|ACTIVE| = 0$, output COL.

8. GOTO Step 1.

Why does this work?

- Every time the GREEDY algorithm stops, we are left with a demanding set that, guaranteed by Lemma 6.6 has more neighbors than its demand. For this set, there are two possibilities. (1) for every subset, there are more neighbors than demand or (2) there exists some subset for which there is more demand than neighbors.

- For case (1), Hall's theorem guarantees a matching, which will be found by MATCH. For case (2), we choose the largest subset for which there is more demand than neighbors. (Note that the remainder of $ACTIVE$ satisfies Hall's theorem because the other subset is maximal, so MATCH takes care of those nodes, which are then removed from $ACTIVE$.) This subset violates the counting condition. Therefore—because every demanding subset of G satisfies the counting condition—this subset is not demanding and there is at least one neighbor that is demanded by only IN or only OUT vertices. This means that the GREEDY subroutine can make progress and either finish or will stop with another demanding $ACTIVE$ set.

Because either GREEDY or MATCH will always continue or finish, we are guaranteed to produce $(3, 2)$-coloring. In other words, we just proved:

Lemma 6.9 *If G is a 4-regular bipartite graph with expansion $\frac{8}{3}^{+}$, then G is $(2n; 3, 2, 4)$-useful.*

Using this lemma in place of Lemma 5.11, we can prove our theorem:

Theorem 6.10 *There exists U_0 and c_0 such that, for all $U \geq U_0$, for all n, there is a $(U, n; s, 4)$-BPDS where $s = \lceil c_0 n^{1/4} U^{3/4} \rceil$.*

Proof: This follows from Lemmas 5.7, 6.9, and 5.13 with $\delta = 1/3$.

∎

6.2 Actual Values and Comparison

To get the actual values for s, we refer back to the proof of Lemma 5.13:

$$c_0 = 2c = (6 - 2\delta)^{1+\delta} \cdot e^{8-\delta}$$

and

$$s \geq c_0 \cdot n^{\frac{\delta}{1+\delta}} \cdot U^{\frac{1}{1+\delta}}$$

We use the same calculation as in Section 5. When $\delta = 1/3$ (expansion $3 - 1/3 = 8/3$).

$$c_0 = \left(\frac{16}{3}\right)^{4/3} \cdot e^{23/3}$$

Using this constant value, Table 7 again compares values of s for different n, this time including the Expansion-$8/3^+$ algorithm with EASY and Expansion-$14/5^+$. For low n-values, EASY still has much better space values, despite a dramatic improvement with the smaller expansion. Table 8, shows s with powers of 2 as n. This is due to the fact that the constant factor for EASY is still much better, even for higher values of n and with the dramatic improvement in the expansion algorithm with a smaller expansion.

n	EASY	$n^{1/6}U^{5/6}$	$n^{1/4}U^{3/4}$
2	$2.634 \cdot 10^5$	$2.306 \cdot 10^{12}$	$3.971 \cdot 10^{11}$
3	$2.634 \cdot 10^5$	$2.467 \cdot 10^{12}$	$4.394 \cdot 10^{11}$
4	$1.718 \cdot 10^{10}$	$2.589 \cdot 10^{12}$	$4.722 \cdot 10^{11}$
5	$1.718 \cdot 10^{10}$	$2.687 \cdot 10^{12}$	$4.993 \cdot 10^{11}$
6	$1.718 \cdot 10^{10}$	$2.770 \cdot 10^{12}$	$5.226 \cdot 10^{11}$
7	$1.718 \cdot 10^{10}$	$2.842 \cdot 10^{12}$	$5.431 \cdot 10^{11}$
8	$1.718 \cdot 10^{10}$	$2.906 \cdot 10^{12}$	$5.615 \cdot 10^{11}$
9	$1.718 \cdot 10^{10}$	$2.963 \cdot 10^{12}$	$5.783 \cdot 10^{11}$
10	$1.718 \cdot 10^{10}$	$3.016 \cdot 10^{12}$	$5.938 \cdot 10^{11}$

Table 7: Values of s using different n for EASY, Expansion-$14/5^+$, and Expansion-$8/3^+$ algorithms. ($U = 2^{32} - 1$, $q = 4$)

n	EASY	$n^{1/6}U^{5/6}$	$n^{1/4}U^{3/4}$
8	$1.718 \cdot 10^{10}$	$2.906 \cdot 10^{12}$	$5.615 \cdot 10^{11}$
16	$1.718 \cdot 10^{10}$	$3.261 \cdot 10^{12}$	$6.678 \cdot 10^{11}$
32	$1.718 \cdot 10^{10}$	$3.661 \cdot 10^{12}$	$7.942 \cdot 10^{11}$
64	$1.718 \cdot 10^{10}$	$4.109 \cdot 10^{12}$	$9.444 \cdot 10^{11}$
128	$1.718 \cdot 10^{10}$	$4.613 \cdot 10^{12}$	$1.123 \cdot 10^{12}$
256	$1.718 \cdot 10^{10}$	$5.178 \cdot 10^{12}$	$1.335 \cdot 10^{12}$
512	$1.718 \cdot 10^{10}$	$5.812 \cdot 10^{12}$	$1.588 \cdot 10^{12}$
1024	$1.718 \cdot 10^{10}$	$6.523 \cdot 10^{12}$	$1.888 \cdot 10^{12}$
2048	$1.718 \cdot 10^{10}$	$7.322 \cdot 10^{12}$	$2.246 \cdot 10^{12}$
4096	$1.718 \cdot 10^{10}$	$8.219 \cdot 10^{12}$	$2.671 \cdot 10^{12}$
8192	$1.718 \cdot 10^{10}$	$9.226 \cdot 10^{12}$	$3.176 \cdot 10^{12}$

Table 8: Values of s using different powers of 2 as n for both EASY, Expansion-$14/5^+$, and Expansion-$8/3^+$ algorithms. ($U = 2^{32} - 1$, $q = 4$)

U	n	EASY	$n^{1/4}U^{3/4}$
$2^{32}-1$	32	$1.718 \cdot 10^{10}$	$7.942 \cdot 10^{11}$
$2^{36}-1$	36	$2.748 \cdot 10^{11}$	$6.543 \cdot 10^{12}$
$2^{40}-1$	40	$4.398 \cdot 10^{12}$	$5.374 \cdot 10^{13}$
$2^{44}-1$	44	$7.036 \cdot 10^{13}$	$4.403 \cdot 10^{14}$
$2^{48}-1$	48	$1.125 \cdot 10^{15}$	$3.600 \cdot 10^{15}$
$2^{52}-1$	52	$1.801 \cdot 10^{16}$	$2.938 \cdot 10^{16}$
$2^{56}-1$	56	$2.882 \cdot 10^{17}$	$2.394 \cdot 10^{17}$
$2^{60}-1$	60	$4.611 \cdot 10^{18}$	$1.948 \cdot 10^{18}$
$2^{64}-1$	64	$7.378 \cdot 10^{19}$	$1.584 \cdot 10^{19}$
$2^{68}-1$	68	$1.180 \cdot 10^{21}$	$1.287 \cdot 10^{20}$
$2^{72}-1$	72	$1.888 \cdot 10^{22}$	$1.044 \cdot 10^{21}$

Table 9: Values of s for EASY and Expansion-$8/3^+$, for increasing U. ($n = \log U, q = 4$)

The expansion bounds, however will eventually win over EASY. For the $q = 4$ case, almost any value of n will cause the EASY bound to be $O(U)$ (as noted in the EASY analysis section). Table 9 gives a comparison between the expansion ($8/3^+$) bound and the EASY bound for increasing U. EASY continues as the better bound until approximately 2^{52} (a reasonable universe).

Still, EASY is a very good upper bound for its constant factor and the fact that it is constructive (or *explicit* in the literature's terminology) and the expansion bounds are not—even though it requires finding a prime number larger than U^{δ}.

7 Lower Bound: If there exists a $(U, n; s, 2)$-BPDS and $n \geq 2$ then $s \geq U$

Next, we present the first of the lower bounds from literature for the 2-query case, originally published in [6]. This proof shows that for *any* 2-query BPDS, the space used to store a set must be at least U (no Ω), no matter how small n may be. Our contribution in this section is the clear and thorough write-up of the proof.

We will use the following key fact from graph theory:

Fact 7.1 *If a graph has more edges than vertices then it must have a cycle.*

Theorem 7.2 *Let $n \geq 2$. In any $(U, n; s, 2)$-BPDS, $s \geq U$.*

Proof:

We will prove this by induction on U.

Base Case $U = 1$ If $s < U$ then $s = 0$. So no queries can be made. Thus if $A = \emptyset$ or $A = \{1\}$, the membership algorithm gives the same answer. This is a contradiction.

Note that $U = 1$, $n \geq 2$ makes sense because $A \subseteq U$ such that $|A| \leq n$. In that case, $|A| \leq 1$.

We do not need the $U = 2$ nor the $U = n$ case; however, we present both for enlightenment.

Case $U = 2$ (For Fun) If $s < U$ then $s = 1$. Map $\emptyset$, $\{1\}$ and $\{2\}$ to how they are stored. Since there is only one bit, two of them map to the same storage. If its $\emptyset$ and $\{1\}$ then the query "$1 \in A$?" will be answered incorrectly. If its $\emptyset$ and $\{2\}$ then the query "$2 \in A$?" will be answered incorrectly. If its $\{1\}$ and $\{2\}$ then query "$2 \in A$?" will be answered incorrectly.

Case $U = n$ (For More Fun) In this case we are storing all subsets of U. If $s < U$ then we can take $s = n - 1$. Map all elements of 2^U to how the $n-1$ bits are set. Since this map had domain of size 2^n and co-domain of size 2^{n-1}, two different sets map to the same setting. Call the sets A, B. Let $a \in A \oplus B$ (so either $a \in A - B$ or $a \in B - A$). The membership query algorithm will make a mistake on the query "$a \in A$?".

Induction Hypothesis Assume that there does not exist a nonadaptive $(U - 1, n; s, 2)$ bit-probe data structure where $s < U - 1$.

Induction Step We can assume $U \geq 2$ from our base cases. Also, assume to reach a contradiction that there exists a $(U, n; s, 2)$-BPDS where $s < U$ (note that for $U - 1$, the induction hypothesis, this is *not* true).

Let $u \in [U]$. To determine if $u \in U$ you make two queries, which we denote a_u and b_u (so the actual queries are to $CELL[a_u]$ and $CELL[b_u]$). Then, you apply some boolean function of two variables to the answers, which we will call $f_u(x, y)$. There are several case of what f_u can be which lead to several cases for our theorem. Most of them are easy. Note that we aim to reach a contradiction in *every* case.

f_u is constant There exists u such that f_u is the constant function.

We take $f_u(x, y) = TRUE$ (FALSE is similar). Let $A = \emptyset$. If you store A in your data structure and ask "$u \in A$?" you get answer TRUE. This is incorrect; a contradiction.

f_u **depends on one input** There exists u such that $f_u(x, y)$ depends on only one of the variables. We take $f_u(x, y) = x$ (the cases $f_u(x, y) = \neg x$, $f_u(x, y) = y$, and $f_u(x, y) = \neg y$ are similar). Hence we have

$$u \in [U] \text{ iff } a_u = 1$$

Consider this claim: If $A \subseteq [U] - \{u\}$ and $|A| \leq n$. When A is stored, $CELL[a_u] = 0$.

To show this is true, assume that when A is stored $CELL[a_u] = 1$. Then the query "$u \in A$?" will be answered YES when it should be NO. Hence, the claim is true.

Using the claim, we can now create a nonadaptive $(s - 1, 2)$ bit-probe data structure where the universe is $[U] - \{u\}$. This will contradict the induction hypothesis. Use the same storage you did for U; but do not use $CELL[a_u]$. If this cell is ever asked about then hardwire the answer 0 into it. Hence it does not count as a cell. Thus we use $s - 1$ cells for universe $U - 1$. Because $s < U$, this contradicts the induction hypothesis.

f_u **depends on both** x **and** y Create an edge-labeled multigraph[6] with vertex set $[s]$ and edge set

$$E = \{(a_u, b_u) : u \in [U] \text{ this edge is labeled } u\}$$

[6]A *multigraph* is a graph where there may be several edges between two nodes, rather than at most one.

It is possible that two different elements of U ask the same two questions—which is why it is a multigraph.

The key to this case is that this graph has s vertices but $U > s$ edges. Hence it must have a cycle. Let the cycle be $(x_1, x_2, \ldots, x_L)$. We will refer to the edges as $u_1, u_2, \ldots, u_L$, where $u_1 = (x_1, x_2)$ and so on. We do not know what L is; it will not matter.

There are several sub-cases:

$f_u = x \wedge y$ **for some** u In this case, the cycle contains an edge labeled u such that $f_u(x, y) = x \wedge y$ We assume $f_{u_1}(x, y) = x \wedge y$ without loss of generality.

We build a set $A \subseteq [U]$, $|A| \leq n$, which will help us get a contradiction.

- Stage 1: $A = \{u_1\}$. $CELL[x_1] = 1$ and $CELL[x_2] = 1$ are forced.
- Stage 2: If $CELL[x_2] = 1$ forces $u_2 \in A$ then we are done: the set $A = \{u_1\}$ will cause a mistake on the query "$u_2 \in A$?". Hence we can assume that $f_{u_2}(x, y)$ is not forced by the choice $x = 1$.

  Now consider another claim: the decision to have $u_2 \notin A$ forces the value of $CELL[x_3]$. Assume not. Say that $A = \{u_1\}$ is stored. Clearly $CELL[x_1] = CELL[x_2] = 1$. What about $CELL[x_3]$. By assumption it is not forced. Hence both the data structures

$$CELL[x_1] = 1, CELL[x_2] = 1, CELL[x_3] = 1$$

$$CELL[x_1] = 1, CELL[x_2] = 1, CELL[x_3] = 0$$

must correctly answer all membership queries for the set $\{u_1\}$.

How do we encode $B = \{u_1, u_2\}$? To get the data structure to answer yes to "$u_1 \in A$?", we must set $CELL[x_1] = CELL[x_2] = 1$. We know that the bit probe queries made to answer "$u_2 \in A$?" are $CELL[x_2]$ and $CELL[x_3]$. But whether you set $CELL[x_3]$ to 0 or 1 you will get the answer NO based on what we know about storing $\{u_1\}$. Hence $\{u_1, u_2\}$ cannot be stored. Therefore, the claim is true: $CELL[x_3]$ is forced to some value when $u_2 \notin A$.

- Stage i: For $i = 3$ to $L-1$ assume inductively that (1) $u_1 \in A$, (2) $u_2, u_3, \ldots, u_{i-1} \notin A$, and (3) $CELL[x_1]$, $CELL[x_2]$, $\ldots$, $CELL[x_i]$ have been forced.

  If the setting of $CELL[x_i]$ forces the status of u_i then we are done: one of the sets $\{u_1, u_i\}$ or $\{u_1\}$ will cause a mistake on the query "$u_i \in A$?". Let b be such that $CELL[x_i]$ be forced to be b. Do not put u_i into A. This forces $CELL[x_{i+1}]$ to some value by similar reasoning to Stage 2.

We now have that if $A = \{u_1\}$ then $CELL[x_1]$, $\ldots$, $CELL[x_L]$ are forced. This forces the status of $\{u_L\}$. Hence one of the sets $\{u_1\}$ or $\{u_1, u_L\}$ will cause a mistake on the query "$u_L \in A$?" and gives our contradiction.

f_u is similar to $x \wedge y$ If the cycle contains an edge labeled u such that $f_u(x, y)$ is one of the following: $x \wedge \neg y, \neg x \wedge y, \neg x \wedge$

$\neg y, x \vee y, x \vee \neg y, \neg x \vee y, \neg x \vee \neg y$, then the logic is just like the $x \wedge y$ case. The key is that there was a setting of $f_u(x, y)$ that forced both x and y. The same is true for these functions.

f_u is $x \oplus y$ or $\neg(x \oplus y)$ The remaining case is where each edge on the cycle has associated f_u of the form $x \oplus y$ or $\neg(x \oplus y)$. Hence we have that there exists $b_1, \ldots, b_L \in \{0, 1\}$ such that

$$\begin{aligned}
A(u_1) = & \quad x_1 + x_2 + b_1 \pmod 2 \\
A(u_2) = & \quad x_2 + x_3 + b_2 \pmod 2 \\
A(u_3) = & \quad x_3 + x_4 + b_3 \pmod 2 \\
\vdots & \quad \vdots \\
A(u_L) = & \quad x_L + x_1 + b_L \pmod 2
\end{aligned}$$

Note the following

- If $A = \emptyset$ then the bits must be set such that, for all $1 \leq i \leq L$, $x_i + x_{i+1} + b_i \equiv 0$. Summing over all $1 \leq i \leq L$ all of the x_i's cancel and you get $\sum_{i=1}^{L} b_i \equiv 0$.
- If $A = \{u_1\}$ then $x_1 + x_2 + b_1 = 1$ but, for all $2 \leq i \leq L$, $x_i + x_{i+1} + b_i \equiv 0$. Summing over all $1 \leq i \leq L$ all of the x_i's cancel and you get $\sum_{i=1}^{L} b_i \equiv 1$.

It cannot be that $\sum_{i=1}^{L} b_i$ is both $\equiv 0$ and $\equiv 1$ so this is a contradiction.

In each case for f_u, we have now arrived at a contradiction. Therefore, the assumption, that there exists a $(U, n; s, 2)$-BPDS where $s < U$ must be false. This concludes the inductive case and our proof.

∎

8 Lower Bound: If there is a $(U, n; s, 3)$-BPDS and $n \geq 4$ then $s \geq c_0 n^{2/3} U^{1/3}$

This section is based on the original proof in [1]. We will first examine the 3-query case and prove the lower bound from literature, then give a new lower bound that holds for all $q \geq 3$, and values for the space required at sample of q and n. This generalization and the resulting space values we will present have not appeared in literature.

We proceed by proving a lemma that links lower bounds for nonadaptive $(U, n; s, q)$-BPDS to a problem in pure combinatorics, Lemma 8.3. This proof demonstrates combinatorial reasoning to prove a lower bound, which is a common technique from the literature ([1], [6]).

Def 8.1 Let $G = (U, s, E)$.

1. If $u \in U$ then $N(u)$ is the set of all neighbors of u.
2. Let $Y \subseteq [s]$. Then

$$\mathrm{ANSBY}(Y) = \{u \in U : N(u) \subseteq Y\}$$

Intuitively, think of $\mathrm{ANSBY}(Y)$ as the members of $[U]$ whose membership could be determined, or answered, given the bits of Y.

Def 8.2 Let $G = (U, s, E)$ and $n \in \mathbb{N}$. G is *n-nice* if,

$$\text{For all } Y \in \bigcup_{i=1}^{n-1} \binom{[s]}{i}$$

$$|\text{ANSBY}(Y)| \le |Y|$$

We will now show that *every* graph associated with a $(U, n; s, q)$-BPDS is n-nice, which will be the key to our lower bound. Specifically, we seek values of s for which the graph (U, s, E) cannot be n-nice.

Lemma 8.3 *If there exists an $(U, n; s, q)$-BPDS, then there is a q-regular bipartite graph $G = (U, s, E)$ that is n-nice.*

Proof: Assume there exists a $(U, n; s, q)$-BPDS. Let $G = (U, s, E)$ be the associated bipartite graph.

Assume, to reach a contradiction, that there exists $Y \in \bigcup_{i=1}^{n-1} \binom{[s]}{i}$, such that $\text{ANSBY}(Y) \ge |Y| + 1$. Let Z be a subset of $\text{ANSBY}(Y)$ of size $|Y| + 1$. Note that $|Z| \le n$. Hence every $A \subseteq Z$ has a representation in the data structure. Note also that the answers given by the data structure for the elements of Z depend only on the bits of Y.

Let $A \subseteq Z$. If A is the set you are trying to store then you will set the bits of Y. Map each such A to the way you set the bits of Y. This is a mapping of a set of size $2^{|Y|+1}$ to a set of size $2^{|Y|}$. Hence two different $A_1, A_2 \subseteq [Z]$ set the bits the exact same way. This is a contradiction because this will cause some membership query to be answered incorrectly.

∎

8.1 The Lower Bound

We will now prove a combinatorial lemma using this n-nice property. This leads directly to the lower bound. Note that there is no Ω in this result.

Lemma 8.4 *Let $s, n, U \in \mathbb{N}$. Let $4 \le n \le U$. If $G = (U, s, E)$ is any n-nice 3-regular bipartite graph then $s \ge n^{2/3}U^{1/3}/4$.*

Proof:

Let $G = (U, s, E)$ be a 3-regular n-nice bipartite graph. We will pick a set $Y \subseteq [s]$, $|Y| = n-1$, at random and find the expected value of ANSBY(Y).

Fix $u \in U$. Note that $|N(u)| = 3$.

$$\Pr(N(u) \subseteq Y) = \frac{\binom{s-3}{n-4}}{\binom{s}{n-1}} = \frac{(s-3)!}{(n-4)!(s-n+1)!}\frac{(n-1)!(s-n+1)!}{s!} =$$

$$\frac{(n-1)(n-2)(n-3)}{s(s-1)(s-2)} \ge \left(\frac{n-3}{s}\right)^3.$$

Hence

$$\Pr(N(u) \subseteq Y) \ge \left(\frac{n-3}{s}\right)^3.$$

Let EX be the expected number of u such that $N(u) \subseteq Y$. By the above calculation

$$EX \ge U\left(\frac{n-3}{s}\right)^3.$$

Since G is n-nice we know that $EX \le n-1$. Hence

$$U\left(\frac{n-3}{s}\right)^3 \le n-1.$$

$$U^{1/3}\frac{n-3}{s} \le (n-1)^{1/3}.$$

$$\frac{n-3}{s} \le \frac{(n-1)^{1/3}}{U^{1/3}}.$$

$$\frac{s}{n-3} \ge \frac{U^{1/3}}{(n-1)^{1/3}}.$$

$$s \ge \frac{U^{1/3}(n-3)}{(n-1)^{1/3}}.$$

$$s \ge \frac{U^{1/3}(n-3)}{(n-1)^{1/3}}.$$

Since $n \ge 4$ we have $n-3 \ge \frac{n}{4}$. Clearly $n-1 \le n$. Hence we have

$$s \ge \frac{U^{1/3}n}{4n^{1/3}} \ge n^{2/3}U^{1/3}/4$$

▮

Theorem 8.5 *If there is a $(U, n; s, 3)$-BPDS and $n \ge 4$, then $s \ge n^{2/3}U^{1/3}/4$.*

Proof: This follows from Lemma 8.3 and Lemma 8.4: A BPDS must have an associated n-nice bipartite graph, and such graphs must have $s \ge$

$n^{2/3}U^{1/3}/4$

▮

8.2 Generalization: If $n \geq q+1$, then $s \geq (U^{1/q}n^{1-1/q})/q$.

Lemma 8.4 can be easily generalized for any value of q, which we will now demonstrate. This generalization will allow us to easily compare this lower bound with the other lower bounds presented in this thesis.

Lemma 8.6 *Let $q < n \leq U$. If $G = (U, s, E)$ is any n-nice q-regular bipartite graph, then $s \geq (U^{1/q}n^{1-1/q})/q$.*

Proof: This proof parallels the proof of Lemma 8.4. We start with G and randomly pick a set $Y \subseteq [s]$ ($|Y| = n-1$). The probability that the neighbors of some $u \in [U]$ are contained in Y is the number of ways to choose Y containing the q neighbors of u divided by the number of ways to choose sets of size $n-1$:

$$P(N(u) \subseteq Y) = \frac{\binom{s-q}{n-q-1}}{\binom{s}{n-1}} = \frac{(s-q)!}{(n-q-1)!(s-n+1)!} \cdot \frac{(n-1)!(s-n+1)!}{s!} =$$

$$\frac{(n-1)(n-2)(n-3)\dots(n-q)}{s(s-1)(s-2)\dots(s-q+1)} \geq \left(\frac{n-q}{s}\right)^q$$

The expected number of u that are in ANSBY(Y) is obtained by multiplying this probability by U:

$$n-1 \geq EX \geq U\left(\frac{n-q}{s}\right)^q$$

$$s^q \geq U\frac{(n-q)^q}{n-1}$$

$$s \geq U^{1/q}\frac{n-q}{(n-1)^{1/q}}$$

Using the assumption (from the Lemma's premise) that $n \geq q+1$, we have $n - q \geq \frac{n}{q}$. Clearly, $n > n-1$. Using both of these facts to simplify, we arrive at our result,

$$s \geq U^{1/q}\frac{n}{q \cdot n^{1/q}}$$

$$s \geq (U^{1/q}n^{1-1/q})/q$$

∎

8.3 Actual Values

To generate an accurate table of results for this lower bound, we take an intermediate step in the proof of Lemma 8.6, before approximating—which made the terms nicer for a stated bound, but less accurate. We use the result,

$$s \geq U^{1/q}\frac{n-q}{(n-1)^{1/q}}$$

The values for this result, shown in Table 10, are much better than the EASY lower bound for most values of n, except values that are near q. A comparison of these space values to the EASY lower bound space values is given in Table 11, with the values for EASY in parentheses. Note that for both tables, $n \geq q$ because of the $(n-q)$ term and the negative results are omitted from the table.

$n \setminus q$	3	4	5	6	7	8	9	10	11	12	13	14	15
4	1128	0											
5	2048	182	0										
6	2852	343	62	0									
7	3579	491	119	30	0								
8	4249	630	172	59	19	0							
9	4877	762	223	86	36	13	0						
10	5471	887	273	112	53	25	10	0					
11	6036	1008	320	138	69	36	19	8	0				
12	6579	1125	366	163	85	48	28	15	7	0			
13	7101	1238	412	187	101	59	36	22	12	6	0		
14	7605	1349	456	211	116	70	45	29	18	11	5	0	
15	8094	1456	499	234	131	81	53	36	24	16	9	5	0
16	8569	1561	541	257	146	92	61	43	30	21	14	9	4
17	9032	1664	583	280	160	102	70	49	36	26	18	12	8
18	9483	1766	623	302	175	113	78	56	41	31	23	16	11
19	9924	1865	664	324	189	123	86	62	47	35	27	20	15
20	10356	1962	703	346	203	133	94	69	52	40	31	24	19

Table 10: Values for this section's lower bounds on s, for different q and n using $U = 2^{32} - 1$. Negative values omitted.

$n \setminus q$	3	4	5	6	7	8	9
4	1128 (1861)	0 (379)					
5	2048 (1861)	182 (379)	0 (154)				
6	2852 (1861)	343 (379)	62 (154)	0 (88)			
7	3579 (1861)	491 (379)	119 (154)	30 (88)	0 (60)		
8	4249 (1861)	630 (379)	172 (154)	59 (88)	19 (60)	0 (46)	
9	4877 (1861)	762 (379)	223 (154)	86 (88)	36 (60)	13 (46)	0 (38)
10	5471 (1861)	887 (379)	273 (154)	112 (88)	53 (60)	25 (46)	10 (38)
11	6036 (1861)	1008 (379)	320 (154)	138 (88)	69 (60)	36 (46)	19 (38)
12	6579 (1861)	1125 (379)	366 (154)	163 (88)	85 (60)	48 (46)	28 (38)
13	7101 (1861)	1238 (379)	412 (154)	187 (88)	101 (60)	59 (46)	36 (38)
14	7605 (1861)	1349 (379)	456 (154)	211 (88)	116 (60)	70 (46)	45 (38)
15	8094 (1861)	1456 (379)	499 (154)	234 (88)	131 (60)	81 (46)	53 (38)
16	8569 (1861)	1561 (379)	541 (154)	257 (88)	146 (60)	92 (46)	61 (38)
17	9032 (1861)	1664 (379)	583 (154)	280 (88)	160 (60)	102 (46)	70 (38)
18	9483 (1861)	1766 (379)	623 (154)	302 (88)	175 (60)	113 (46)	78 (38)
19	9924 (1861)	1865 (379)	664 (154)	324 (88)	189 (60)	123 (46)	86 (38)
20	10356 (1861)	1962 (379)	703 (154)	346 (88)	203 (60)	133 (46)	94 (38)

Table 11: Space values for this section's lower bound compared to values for the EASY lower bound (in parentheses), for different q and n using $U = 2^{32}-1$. Negative values omitted.

The space lower bound from this method depends on n, which is the edge over the EASY lower bound. EASY depends on M, which is based on n, but doesn't increase as n increases. Intuitively, it makes sense that the space required should depend on the number of items being stored and that this generalized upper bound is closer.

9 Lower Bound: If there is a $(U, n; s, 3)$-BPDS and $n \geq 16 \log U$ then $s \geq \Omega\left(\frac{n^{1/2}U^{1/2}}{(\log U)^{1/2}}\right)$

Our next lower bound is also from [1], but presented here in more detail. Our method to prove this lower bound is, again, to examine associated bipartite graphs, and find the values of s for which (U, s, E) cannot be n-nice. To do this, we first prove a purely graph-theoretic result, Lemma 9.4. This bound is asymptotically better than the bound in 8 and we present comparison values at the end of this section.

Our contribution is the thorough write-up of these results and the value comparison between the lower bounds.

9.1 Graph-Theoretic Results

Our first lemma is folklore–it is cited as "well known" in [1]–we include a proof here for completeness. This proof is based on a comment we found at [2].

Lemma 9.1 *Any graph with s vertices and at least $2s$ edges contains a cycle no longer than $2 \log s$*

Note 9.2 It is important to know what exactly Lemma 9.1 states. We know from Lemma 7.1 that any such graph must have at least one cycle; in contrast, this lemma states that the graph must have at least one *small* cycle (where *small* means $\leq 2 \log s$).

Proof: We proceed by induction on s.

Base Case Begin[7] at $s = 5$. The only graph with 5 vertices and 10 edges is the complete graph (K_5), so there are many small cycles of length $3 < 2 \log 5 \approx 4.64$

Induction Hypothesis Assume that any graph with $n-1$ vertices and $\geq 2(n-1)$ edges contains a cycle of length $\leq 2\log(n-1)$.

Inductive Case (1) Let G be a graph of size n with $\geq 2n$ edges. If G contains a vertex with degree 1 or 2, then we can form G' by omitting that vertex and its incident edges (no more than 2). G' then has $n-1$ vertices and $\geq 2n-2 = 2(n-1)$ edges, accounting for those that were removed. This means that the induction hypothesis applies to G' and that there is a small cycle in G' that must also exist in G.

Inductive Case (2) If G does not have a vertex of degree 1 or 2, then each vertex has at least 3 edges. Consider a breadth-first search method to build a tree on this graph. Start at any vertex and add as its children each of its neighbors. Keep adding neighbors until a previously-visited node (a cycle!) is found. How big can this cycle be? In the worst case, we would exhaust all of the nodes in the graph this way, forming a tree with all s vertices. We know, however, that each node in the tree has at least two children because it has degree ≥ 3. So the maximum depth of the tree is $\log n$ and the cycle is no longer than $2\log n$.

∎

Lemma 9.3 *Any graph with s vertices and more than $3s$ edges contains a set of $k \leq 4\log s$ vertices which spans at least $k+1$ edges.*

[7]Because smaller values of s result in graphs that cannot contain $\geq 2s$ edges!

Proof: Let G be a graph with s vertices and $\geq 3s$ edges. By Lemma 9.1, there is a cycle of length $\leq 2\log s$. Take such a cycle and form a new graph by omitting its edges. Keep repeating this process until the graph formed by this process no longer has at least $2s$ edges—no longer guaranteed by Lemma 9.1 to have a small cycle.

At this point, we have a set of edge-disjoint cycles and a graph, G' with less than $2s$ edges. Because G' has less than $2s$ edges, we know that the set of cycles contains more than s edges, so two of those cycles must share a common vertex. Let k_1 and k_2 be the number of edges in those cycles. The number of vertices in each cycle are also k_1 and k_2 (which are both $\leq 2\log s$), but the number of vertices in their union, k, is $\leq k_1 + k_2 - 1$ because they share at least one vertex. The number of edges in their union is $k_1 + k_2$ because both cycles are edge-disjoint. So the union of these two cycles forms a subgraph with $k \leq 4\log s$ vertices with at least $k+1$ edges, as in the statement of the lemma. ▮

Lemma 9.4 *For $s \geq n \geq 16\log s$, any graph with s vertices and at least $3s + n/2$ edges contains a set of $k \leq n/2$ vertices that spans at least $k + \frac{n}{16\log s}$ edges.*

Proof: Let G be a graph with s vertices and $\geq 3s + n/2$ edges. By Lemma 9.3, there is a set, X_1 of k_1 vertices that spans more than $k_1 + 1$ edges. Form a new graph by omitting its edges, and repeat. Perform this process $\frac{n}{16\log s}$ times.

Let $X = \bigcup_{i=1}^{n/16\log s} X_i$. This set has k vertices, where

$$k \leq (4\log s) \cdot \frac{n}{16\log s} = \frac{n}{4} \leq \frac{n}{2}$$

Because each set of vertices spans at least $k_i + 1$ edges, and all of the edge sets are disjoint—because in each iteration, we removed them from the construction of the next set—the number of edges spanned by X is

$$\begin{aligned} \sum_{i=1}^{n/16\log s} k_i + 1 &= \sum_{i=1}^{n/16\log s} k_i + \sum_{i=1}^{n/16\log s} 1 \\ &\geq k + \frac{n}{16\log s} \end{aligned}$$

Therefore, X is a set as in the statement of the lemma.

Note 9.5 We know that Lemma 9.3 applies during each iteration, because each set X_i has no more than $4\log s$ edges. $4\log s \cdot \frac{n}{16\log s} = \frac{n}{4}$, and we know (by the premise of this lemma) that G contained $\geq 3s + n/2$ edges. Thus, at the end and during each iteration, there are at least $3s + n/4$ edges and we can apply Lemma 9.3.

∎

Note 9.6 The three lemmas in this section also apply to multi-graphs—graphs where the set of edges, E, can contain duplicates. The proofs are the same because we never relied on the fact that each edge was unique, only that each edge was never used twice. It is in the context of multi-graphs that we will use these lemmas in Section 9.2

9.2 The Lower Bound: Graph Limits of n-nice

We now apply the graph lemmas to get our result by showing that for certain values of s, there are no n-nice graphs.

Theorem 9.7 *There exists c_0 such that, for all U, if there is an $(U, n; s, 3)$-BPDS with $n \geq 16 \log U$, then*

$$s \geq c_0 \frac{n^{1/2} U^{1/2}}{(\log U)^{1/2}}$$

Proof: Let $G = (U, s, E)$ be the corresponding bipartite graph to a $(U, n; s, 3)$-BPDS where $s \geq n \geq 16 \log U$ Let $R \subseteq S$ be the $\frac{n}{17 \log s}$ vertices of largest degree in S. The sum of the degrees of each $z \in R$ is greater than $\frac{n}{17 \log s} \cdot \frac{3U}{s}$, because $\frac{3U}{s}$ is the average degree of the members of $[s]$ and the members of R have the largest degrees.

Let G' be the multi-graph $([s], E')$. E' is the set of edges formed by creating an edge (x, y) for every $u \in U$ such that the third $z \in N(u)$ is in R. Note that there is a one-to-one relationship between edges in E' and members of U. Formally,

$$E' = \{(x, y) : (\exists_1 u \in U)\,(z \in R \wedge \{(u, x), (u, y), (u, z)\} \subset E)\}$$

Because there is an edge in E' for each $u \in U$ connected to a $z \in R$, the set E' has at least $\frac{3U \cdot n}{17 s \log s} \cdot \frac{1}{3}$ edges. This is equal to the lower bound on the sum of the degrees from above, divided by 3 to account for the situation where x, y, and z (or two of them) are in R.

If the number of edges exceeds $3s + n/2$, we can apply Lemma 9.4 to find a set, T, of $k \leq n/2$ vertices that spans at least $k + \frac{n}{16 \log s}$ edges of E'.

Consider the set $R \cup T$. The size of $R \cup T$ is $g \leq k + \frac{n}{17 \log s} \leq n$. The number of $u \in U$ such that $N(u) \in R \cup T$ is at least $k + \frac{n}{16 \log s} \geq g$, because every edge (x, y) of the set of edges spanned by T ($\subseteq E'$) corresponds to a $u \in U$ such that $N(u) = x, y, z$ and $z \in R$. Therefore,

$$|ANSBY(R \cup T)| > |R \cup T|$$

This contradicts the fact that G is n-nice, so we conclude that

$$\begin{aligned} \frac{U \cdot n}{17 s \log s} &\leq 3s + \frac{n}{2} \leq \frac{7s}{2} \\ \frac{2U \cdot n}{119 \log s} &\leq s^2 \end{aligned}$$

The resulting lower bound is reached using $c_0 = \sqrt{2/119}$ and $s \leq U$ (so $\log s \leq \log U$) to simplify:

$$s \geq c_0 \cdot \frac{n^{1/2} U^{1/2}}{(\log U)^{1/2}}$$

■

9.3 Actual Values and Comparison

The technique from this section also gives a better lower bound than EASY due to the fact that it depends on n. Table 12 shows the actual space values for this lower bound technique. It also gives the 3-query results from Sections 3 and 8 for comparison. The generalized lower bound from Section 8 is the clear winner for our choice of U. The technique in this section gives better bounds than the last section when n is smaller and U is larger (in comparison to the probabilistic technique), but this is limited by the necessary assumption that $n \geq 16 \log U$. For $U = 2^{32} - 1$, n is greater than 512 and the combinatorial technique, which depends more on n and less on U, already gives a better bound.

The graph limits of n-nice, however, do give a better asymptotic bound in some cases. For higher values of U, where does this begin to take effect? Table 13 gives the lower bounds for increasing values of U from $2^{32} - 1$ to $2^{64} - 1$. For each row, we use the minimum $n = 16 \cdot \log U$. The table shows that for large values of U, approximately 2^{44}, the lower bound from this section is a better bound. This also appears to be a reasonable universe. We chose to base most comparisons on the universe $U = 2^{32} - 1$ based on common integer representations, but $2^{64} - 1$ is also a commonly used integer representation on newer 64-bit architectures.

If we choose an n closer to U, for instance $n = U - 16 \cdot \log U$, the trend reverses and the lower bound from Section8 is again the better bound. Table 14 shows the same content as Table 13, but uses $n = U - 16 \cdot \log U$ instead of $16 \cdot \log U$.

n	EASY	$U^{1/q}n^{1-1/q}/q$	$\frac{n^{1/2}U^{1/2}}{(\log U)^{1/2}}$
512	1861	103490	33985
1024	1861	164711	48062
2048	1861	261803	67970
4096	1861	415857	96123
8192	1861	660347	135939

Table 12: Values of s for each lower bound technique and different n. ($U = 2^{32} - 1$, $q = 3$)

U	n	$U^{1/q}n^{1-1/q}/q$	$\frac{n^{1/2}U^{1/2}}{(\log U)^{1/2}}$
$2^{32}-1$	512	$1.034 \cdot 10^5$	$3.398 \cdot 10^4$
$2^{36}-1$	576	$2.822 \cdot 10^5$	$1.359 \cdot 10^5$
$2^{40}-1$	640	$7.633 \cdot 10^5$	$5.437 \cdot 10^5$
$2^{44}-1$	704	$2.050 \cdot 10^6$	$2.175 \cdot 10^6$
$2^{48}-1$	768	$5.477 \cdot 10^6$	$8.700 \cdot 10^6$
$2^{52}-1$	832	$1.456 \cdot 10^7$	$3.480 \cdot 10^7$
$2^{56}-1$	896	$3.856 \cdot 10^7$	$1.392 \cdot 10^8$
$2^{60}-1$	960	$1.017 \cdot 10^8$	$5.568 \cdot 10^8$
$2^{64}-1$	1024	$2.677 \cdot 10^8$	$2.227 \cdot 10^9$

Table 13: Comparison of lower bounds for Sections 8 and 9, for increasing U values. $n = \lceil 16 \cdot \log U \rceil, q = 3$.

U	n	$U^{1/q}n^{1-1/q}/q$	$\frac{n^{1/2}U^{1/2}}{(\log U)^{1/2}}$
$2^{32}-1$	$4.294 \cdot 10^9$	$4.294 \cdot 10^9$	$9.842 \cdot 10^7$
$2^{36}-1$	$6.871 \cdot 10^{10}$	$6.871 \cdot 10^{10}$	$1.484 \cdot 10^9$
$2^{40}-1$	$1.099 \cdot 10^{12}$	$1.099 \cdot 10^{12}$	$2.253 \cdot 10^{10}$
$2^{44}-1$	$1.759 \cdot 10^{13}$	$1.759 \cdot 10^{13}$	$3.438 \cdot 10^{11}$
$2^{48}-1$	$2.814 \cdot 10^{14}$	$2.814 \cdot 10^{14}$	$5.266 \cdot 10^{12}$
$2^{52}-1$	$4.503 \cdot 10^{15}$	$4.503 \cdot 10^{15}$	$8.096 \cdot 10^{13}$
$2^{56}-1$	$7.205 \cdot 10^{16}$	$7.205 \cdot 10^{16}$	$1.248 \cdot 10^{15}$
$2^{60}-1$	$1.152 \cdot 10^{18}$	$1.152 \cdot 10^{18}$	$1.929 \cdot 10^{16}$
$2^{64}-1$	$1.844 \cdot 10^{19}$	$1.844 \cdot 10^{19}$	$2.989 \cdot 10^{17}$

Table 14: Comparison of lower bounds for Sections 8 and 9, for increasing U values. $n = U - \lceil 16 \cdot \log U \rceil, q = 3$.

10 Conclusion

We have studied the bit probe data structure and summarized the deterministic, non-adaptive results from literature. We introduced the bit probe data structure with two new techniques, the EASY upper and EASY lower bounds. We also presented a new probabilistic adaptation of the EASY upper bound.

We continued by presenting complete proofs of the graph expansion upper bounds—both 14/5 and 8/3 versions. Our version is more detailed that those in literature, and includes a correction to the final bound, given in [1] as $O(n^{1/3}U^{2/3})$. We added a new comparison between the known and new upper bounds with actual values for required space. We found that the graph expansion algorithms require more space than the EASY algorithm for low U, but have better space efficiency in most cases. Also, EASY is the only constructive algorithm that works for any q.

Lastly, we presented detailed proofs of lower bounds from literature: the 2-query lower bound from [6], and more complete proofs of the n-nice combinatorial lower bound from [1] and the graph-theory lower bound from [1]. We also included a new generalization of the combinatorial lower bound and new concrete comparisons and constant analysis between the known and new lower bounds. We found that the general combinatorial lower bound in Section 8 is better than the EASY lower bound, except for where q and n are close or when $n < q$. We also found that the graph-theory lower bound outperformed the combinatorial lower bound when n is not near U and when U is sufficiently large, however, this result is only for 3 queries.

References

[1] N. Alon and U. Feige. On the power of two, three and four probes. *Manuscript*, 2008.

[2] Aravind. A "well known" theorem (comments), November 2008. `https://www.blogger.com/comment.g?blogID=3722233&postID=4951420787206539360`.

[3] R. C. Baker, G. Harman, and J. Pintz. The difference between consecutive primes. II. *Proc. London Math. Soc. (3)*, 83(3):532–562, 2001. `http://plms.oxfordjournals.org/cgi/reprint/83/3/532`.

[4] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[5] D. R. Heath-Brown. Differences between consecutive primes. *Jahresber. Deutsch. Math.-Verein.*, 90(2):71–89, 1988.

[6] H. Huhrman, P. Milterson, J. Radhakrishnan, and S. Venkatesh. Are bit vectors optimal. *SIAM Journal on Computing*, 31:1723–1744, 2002. `http://www.daimi.au.dk/~bromille/Papers/index.html`.

[7] I. Simon. Bipartite matching. *Manuscript*. `http://www.mcs.csuhayward.edu/~simon/handouts/4245/hall.html`.