Lecture 5: k-wise Independent Hashing and Applications

Topics in Complexity Theory and Pseudorandomness (Spring 2013)
Rutgers University
Swastik Kopparty
Scribes: Yun Kuen Cheung, Aleksandar Nikolov

1 Overview

In this lecture, we will introduce k-wise independence and k-wise independent hashing. We will discuss some of their applications, including derandomizing approximation algorithms for MAX-CUT and MAX-E3-SAT, data structures for the set membership problem, and constructing polynomial size AC^0 circuits for threshold functions with a polylogarithmic threshold.

2 k-wise Independence

Definition 1. *Let $X_1, X_2, \cdots, X_n$ be a collection of random variables, with respective ranges $A_1, A_2, \cdots, A_n$. Assume $|A_j|$ is finite for all $1 \leq j \leq n$. The random variables are* independent *if for any $a_i \in A_i$,*

$$Pr\left[(X_1 = a_1) \wedge (X_2 = a_2) \wedge \cdots \wedge (X_n = a_n)\right] = \prod_{j=1}^{n} Pr\left[X_j = a_j\right].$$

For $k \geq 2$, the random variables are k-wise independent if for any $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ and any $a_{i_1} \in A_{i_1}$, $a_{i_2} \in A_{i_2}$, ..., $a_{i_k} \in A_{i_k}$,

$$Pr\left[X_{i_1} = a_{i_1} \wedge X_{i_2} = a_{i_2} \wedge \cdots \wedge X_{i_k} = a_{i_k}\right] = \prod_{j=1}^{k} Pr\left[X_{i_j} = a_{i_j}\right].$$

2-wise independence is also called pairwise independence.

If the random variables $X_1, X_2, \cdots, X_n$ are independent, they are k-wise independent for any k. However, the converse is not true as shown by the following example. Also, if $k_1 > k_2$, k_1-wise independence implies k_2-wise independence.

Example 2. *Let X_1, X_2 be uniform independent random variables taking values in $\{0, 1\}$. Let $X_3 = X_1 \oplus X_2$. It is easy to check that $\{X_1, X_2, X_3\}$ is pairwise independent, but not independent.*

3 Applications to Derandomization

We will discuss two classical examples of randomized algorithms, each using $\Omega(n)$ independent random bits. However, the analysis of these randomized algorithms only requires the weaker k-wise

independence for constant k. As we will see in Section 4, $k \lg n$ independent random bits suffice to generate n k-wise independent random bits. The logarithmic improvement in the number of independent random bits needed allows us to convert the randomized algorithms into polynomial time deterministic algorithms.

Example 3. *In the MAX-CUT problem we are given $G = (V, E)$ as input, and our objective is to find a partition of $V = V_0 \cup V_1$ such that the size of $E(V_0, V_1) = \{e = \{u, v\} \in E \mid (u \in V_0, v \in V_1) \vee (u \in V_1, v \in V_0)\}$ is maximized. MAX-CUT is known to be NP-hard. There is a simple randomized approximation algorithm: for each $v \in V$, put it in V_0 and V_1 randomly, independently and uniformly. Then*

$$\begin{aligned}\mathbb{E}\left[|E(V_0, V_1)|\right] &= \sum_{e=\{u,v\}\in E} \left(Pr\left[(u \in V_0) \wedge (v \in V_1)\right] + Pr\left[(u \in V_1) \wedge (v \in V_0)\right]\right) \\ &= \sum_{e=\{u,v\}\in E} \frac{1}{2} = \frac{1}{2}|E| \geq \frac{1}{2} \max_{V_0, V_1} |E(V_0, V_1)|.\end{aligned}$$

Hence, this randomized algorithm achieves a $\frac{1}{2}$-approximation in expectation. In the above calculation, we don't really need each $v \in V$ to be put into V_0 and V_1 independently, we only need pairwise independence. It is known that $\lg n$ bits suffice to generating n pairwise independent random bits (see Example 5). By exhausting all $2^{\lg n} = n$ possibilities of the pairwise independent random bits, and choosing the one which gives the largest $|E(V_0, V_1)|$, we have a deterministic $\frac{1}{2}$-approximation to MAX-CUT.

Remark. There is another deterministic greedy algorithm that achieves $\frac{1}{2}$-approximation.

Remark. The best known approximation ratio is 0.878, achieved by Goemans and Williamson using semi-definite programming and randomized rounding. Their algorithm was derandomized by Mahajan and Ramesh.

Example 4. *A clause is a disjunction of boolean literals, e.g. $x_1 \vee \overline{x_4} \vee x_6$. Given m clauses, each clause having three distinct variables, MAX-E3-SAT is the problem to find an assignment to the variables such that the number of clauses satisfied is maximized. MAX-E3-SAT is known to be NP-hard. There is a simple randomized algorithm: for each variable, set it to TRUE or FALSE randomly, independently and uniformly. Then*

$$\begin{aligned}\mathbb{E}\left[\textit{number of clauses satisfied}\right] &= \sum_c Pr\left[\textit{clause } c \textit{ is satisfied}\right] \\ &= \sum_c \frac{7}{8} = \frac{7}{8}m \geq \frac{7}{8} \max_x (\textit{number of clauses satisfied by assignment } x).\end{aligned}$$

Hence the randomized algorithm achieves a $\frac{7}{8}$-approximation in expectation. Here we only need 3-wise independence. It is known that $2 \lg n$ bits suffice to generating n 3-wise independent random bits. By exhausting all $2^{2 \lg n} = n^2$ possibilities of the 3-wise independent random bits, and choosing the one which gives the maximum number of satisfied clauses, we have a deterministic $\frac{7}{8}$-approximation to MAX-E3-SAT.

4 Generating k-wise Independent Random Variables

In this section we will give three examples of generating k-wise independent random variables with simple families of hash functions.

Example 5. *Let $x_1, x_2, \cdots, x_t$ be uniformly random bits. For each non-empty subset S of $[t]$, define $x_S = \oplus_{i \in S} x_i$. We claim that the collection of random bits $\{x_S\}$ are pairwise independent: given distinct S_1, S_2, WLOG we may assume $S_1 \not\subset S_2$, i.e. $T := S_1 \setminus S_2$ is not empty; then it is easy to see that*

$$\begin{aligned} Pr[x_{S_1} = a \wedge x_{S_2} = b] = Pr[x_{S_2} = b \wedge x_T = a \oplus x_{S_1 \cap S_2}] &= Pr[x_{S_2} = b] \times Pr[x_T = a \oplus x_{S_1 \cap S_2}] \\ &= (1/2)^2 \\ &= Pr[x_{S_1} = a] \times Pr[x_{S_2} = b]\,. \end{aligned}$$

In this example, we need t uniform random bits to generate $2^t - 1$ pairwise independent random bits.

Example 6. *Let $\mathbb{F} = \mathbb{F}_n$ be a finite field of n elements. Pick $c, d \in \mathbb{F}$ uniformly at random. Define $X_\alpha = c \cdot \alpha + d$, for all $\alpha \in \mathbb{F}$. Note that X_α is uniformly distributed in $\mathbb{F}$. We claim that for $\alpha \neq \beta$, X_α and X_β are pairwise independent:*

$$\begin{aligned} Pr[X_\alpha = a \wedge X_\beta = b] = Pr_{c,d}[c \cdot \alpha + d = a \wedge c \cdot \beta + d = b] &= Pr\left[c = \frac{a-b}{\alpha - \beta} \wedge d = \frac{a\beta - b\alpha}{\alpha - \beta}\right] \\ &= (1/|\mathbb{F}|)^2 \\ &= Pr[X_\alpha = a] \times Pr[X_\beta = b]\,. \end{aligned}$$

In this example, we need $2 \log |\mathbb{F}|$ uniform random bits for picking c, d to generate $|\mathbb{F}|$ pairwise independent random variables, each with range in $\mathbb{F}$.

Example 7. *This is a generalization of Example 6 to generate k-wise random variables for arbitrary k. Let $\mathbb{F} = \mathbb{F}_n$ be a finite field of n elements. Pick $c_0, c_1, \cdots, c_{k-1} \in \mathbb{F}$ uniformly at random. Define $X_\alpha = \sum_{i=0}^{k-1} c_i \alpha^i$, for all $\alpha \in \mathbb{F}$. Note that X_α is uniformly distributed in $\mathbb{F}$. Given $a_1, a_2, \cdots, a_k \in \mathbb{F}$ and distinct $\alpha_1, \alpha_2, \cdots, \alpha_k$, by Lagrange interpolation we know that there exists unique $c_0, c_1, \cdots, c_{k-1} \in \mathbb{F}$ such that $X_{\alpha_1} = a_1, X_{\alpha_2} = a_2, \cdots, X_{\alpha_k} = a_k$. Hence*

$$Pr[X_{\alpha_1} = a_1 \wedge X_{\alpha_2} = a_2 \wedge \cdots \wedge X_{\alpha_k} = a_k] = (1/|\mathbb{F}|)^k = \prod_{i=1}^{k} Pr[X_{\alpha_i} = a_i]\,.$$

In this example, we need $k \log |\mathbb{F}|$ uniform random bits for picking $c_0, c_1, \cdots, c_{k-1}$ to generate $|\mathbb{F}|$ k-wise independent random variables, each with range in $\mathbb{F}$.

Remark. It is not difficult to see that if $|\mathbb{F}| = 2^j$ and $0 \leq i \leq j$, Examples 6 and 7 allow us to use kj uniform random bits to generate 2^j uniformly random elements of $[2^i]$ which are k-wise independent.

5 Set Membership Problem

In this section we introduce the bit probe and cell probe models for static data structures and discuss applications of k-wise independence to constructing efficient data structures.

5.1 Static Data Structures

The set membership problem is defined as follows. Let $[m]$ be the *universe*, and let the input be $S \subset [m]$ with $|S| = n$. Based on S, one writes data structure $f(S)$ to memory. Based only on $f(S)$, a query algorithm must be able to answer, for all $i \in [m]$, "Is $i \in S$?" using few probes to memory.

There are two models of probing:

1. Bit Probe Model (BPM): the *space* complexity is measured in the number of bits of $f(S)$ and the *query* complexity is measured in the number of bits accessed (probed);
2. Cell Probe Model (CPM): the data structure $f(S)$ is written in blocks, or *cells*, where each cell consists of w bits; we usually assume $w = \Omega(\log m)$; the *space* complexity is measured in the number of cells of $f(S)$ and the *query* complexity is measured in the number of cells accessed (probed).

Remark. Since we are not bounding the complexity of computing $f(S)$, randomization does not help for computing the data structure: any randomized f is dominated by an f' that fixes the random bits to the optimal choice. However, we will see later that randomized query algorithms can have lower complexity than deterministic ones.

Assume $m = n^{1+\Omega(1)}$. The smallest space complexity possible is $\log \binom{m}{n} \sim \Omega(n \log m)$ bits, or $\Omega(n)$ cells, as a data structure (together with a query algorithm) can be used to reconstruct S, and, therefore, must distinguish between all size n subsets of $[m]$.

A simple solution in the CPM with optimal space complexity is to write all elements of S to memory in sorted order. Then the query algorithm can answer a query for i by doing binary search on the data structure. The query complexity is $O(\log n)$ cell probes (similar complexity is achieved in the dynamic setting by a binary search tree). The next subsection describes a data structure which works for the static case, in which the smallest memory possible is achieved and takes only $O(1)$ probes in CPM.

5.2 Set Membership in Linear Space and $O(1)$ Cell Probes

We will present a data structure due to Fredman, Komlos, and Szemeredi that supports set membership queries with constant query complexity and linear space in CPM. The data structure makes crucial use of 2-wise independent distributions. The complexity of the data structure is optimal up to constants: as we argued, any data structure which answers all membership queries correctly needs to store $\Omega(n)$ cells (in the regime $m = n^{1+\Omega(1)}$) and clearly needs to probe at least a single cell per query.

A natural strategy to solve the set membership problem is hashing: map the input to a set of l cells using a one to one function. More precisely, we hope to find a family of functions $\mathcal{H} = \{h : [m] \to [l]\}$, where l is as small as possible and for any $S \subseteq [m]$ there exists an $h \in \mathcal{H}$ such that the restriction $h|_S : S \to [l]$ of h to S is one to one. Given such a family we can construct a data structure as follows: we write $S \cap h^{-1}(1)$ in the first cell, $S \cap h^{-1}(2)$ in the second, etc., and, $S \cap h^{-1}(l)$ in the l-th cell; we also write down an identifier of the function h which was used. Given a query element i, the query algorithm reads the identifier of h and probes the j-th cell for $j = h(i)$. If that cell stores i, then i is in S, otherwise it is not. Obviously, if $\mathcal{H}$ is the set of all functions

from m to n, then $\mathcal{H}$ satisfies the desired property: for any S there is an $h \in \mathcal{H}$ which is one to one on S. However, such an $\mathcal{H}$ is too big: it has size m^n, so we need $\Omega(n)$ cells to identify a function $h \in \mathcal{H}$ and the query algorithm needs to probe $\Omega(n)$ cells to identify which h was used. So it is also important to find a family $\mathcal{H}$ of polynomial size.

If we allow l to be n^2, then 2-wise independence provides a family with the above properties. Let us first define what we mean by a pairwise independent hash family.

Definition 8. *A pairwise-independent hash family is a set of functions $\mathcal{H} = \{h : [m] \to [l]\}$ such that for all $a, b \in [m]$ and all $c, d \in [l]$ we have $\Pr_h[h(a) = c \wedge h(b) = d] = 1/l^2$, where the probability is taken over choosing a uniformly random $h \in \mathcal{H}$.*

Pairwise independent hashing is simply a different viewpoint on 2-wise independent distributions. When h is chosen at random from $\mathcal{H}$, $h(1), \ldots, h(m)$ are m 2-wise independent random variables taking values in $[l]$. Similarly, any m 2-wise independent random variables $x_1, \ldots, x_m$ taking values in $[l]$ give a pairwise independent hash family: we can associate a hash function h to every choice of random bits and set $h(a) = x_a$ for all $a \in [m]$. Recalling our construction of 2-wise independent distributions, we can construct a pairwise-independent hash family as $\mathcal{H} = \{ax + b : a, b \in \mathbb{F}\}$ for a field $\mathbb{F}$ of size l. Such a family $\mathcal{H}$ has size l^2.

A simple scheme. Let us analyze the data structure we described above when $\mathcal{H}$ is a pairwise independent family. First, we prove that a pairwise-independent $\mathcal{H}$ has a one to one function for every S.

Lemma 9. *Let $\mathcal{H} = \{h : [m] \to [l]\}$ be a pairwise independent hash family and let $l \geq n^2$. Then for any $S \subseteq [m]$ of size $|S| = n$, there exists an $h \in \mathcal{H}$ such that the restriction $h|_S$ is a one to one function from S to $[l]$.*

Proof. We will use the probabilistic method. Let us choose h uniformly at random from $\mathcal{H}$. For $a, b \in S$, let $X_{a,b}$ be the indicator random variable that takes value 1 if and only if $h(a) = h(b)$. I.e. $X_{a,b}$ is 1 if and only if h makes a and b collide. We can write the expected number of collisions as

$$\mathbb{E}\left[\sum_{a<b} X_{a,b}\right] = \sum_{a<b} \mathbb{E}[X_{a,b}] = \frac{n(n-1)}{2l}.$$

Above we used the fact that, by the pairwise independence of $\mathcal{H}$,

$$\Pr[X_{a,b} = 1] = \sum_{c \in [l]} \Pr[h(a) = h(b) = c] = \frac{1}{l}.$$

Since $l \geq n^2$, the expected number of collisions is less than $1/2$, and therefore there exists an $h^* \in \mathcal{H}$ that causes less than $1/2$ collisions of elements of S. But the number of collisions is an integer, therefore $h^*|_S$ is one to one. □

Lemma 9 gives us a data structure with $O(n^2)$ cells and $O(1)$ query time: as discussed above, given S, we find an h such that $h|_S$ is one to one, and we write down $h^{-1}(1) \cap S, \ldots, h^{-1}(l) \cap S$ as well as an identifier of h, for example a and b if we use $\mathcal{H} = \{ax + b\}$. Then a query algorithm reads the identifier of h and, given a query i, reads $h^{-1}(j) \cap S$ where $h(i) = j$. The algorithms answers "yes" if and only if the result is i. Since the identifier of h can be written in constant number of cells, and h is one to one, evaluating a query takes constant time.

The real scheme. We are now ready to present the full scheme of Fredman, Komlos, and Szemeredi. The disadvantage of the simple scheme above is that it uses suboptimal space: a quadratic number of cells, when only a linear number is necessary. As a first step, let us see how many collisions we have when we hash into a set of linear size.

Lemma 10. *Let $\mathcal{H} = \{h : [m] \to [l]\}$ be a pairwise independent hash family and let $l \geq n$. Then for any $S \subseteq [m]$ of size $|S| = n$, there exists an $h \in \mathcal{H}$ which causes less than $n/2$ elements of S to collide.*

The proof of Lemma 10 is identical to the proof of Lemma 9. For input S, let us pick an h that satisfies the conclusion of Lemma 10, i.e causes less than $n/2$ collisions of elements of S. Then in the data structure we write:

- for each $j \in [l]$:
  - the size of the inverse $n_j = |h^{-1}(j) \cap S|$;
  - a pointer to an instance of the *simple scheme* with input $h^{-1}(j) \cap S$;
- an identifier of h.

On query i, the query algorithm reads h and reads the cells corresponding to $j = h(i)$; then it accesses the corresponding instance of the simple scheme and answers the query. Since the simple scheme supports answering queries with a constant number of probes, and the overhead to access the right instance of the simple scheme requires a constant number of probes, the query complexity is $O(1)$ probes. However, we need to analyze the space complexity of the data structure. Notice that the total number of cells used is up to a constant factor equal to $n + n_1^2 + \ldots + n_n^2$. The following lemma then completes the analysis.

Lemma 11. *Let $h : S \to [l]$ be such that there are at most $n/2$ collisions of elements of S. Then we have*

$$\sum_{j=1}^{l} n_j^2 \leq 2n$$

Proof. The number of elements a, b that collide at j, i.e. $|\{(a, b) : a \neq b \wedge h(a) = h(b) = j\}|$, is $\binom{n_j}{2}$. Since the total number of collisions is assumed to be at most $n/2$, we have

$$\frac{n}{2} \geq \sum_{j=1}^{n} \binom{n_j}{2} = \sum_{j=1}^{n} \frac{n_j^2}{2} - \frac{n_j}{2} = \frac{1}{2} \sum_{j=1}^{l} n_j^2 - \frac{n}{2}.$$

The lemma follows after rearranging the terms. □

Putting everything together, we get the following theorem.

Theorem 12. *There exists a data structure for the set membership problem in the cell probe model that uses $O(n)$ cells of space and answers each membership query using $O(1)$ cell probes.*

6 Hashing and Computing Threshold Functions in AC^0

Threshold functions are a family of boolean function $\{f_t : \{0,1\}^n \to \{0,1\} : 0 \le t \le n+1\}$, where $f_t(x) = 1$ if and only the hamming weight of x satisfies $\mathrm{wt}(x) \ge t$. The threshold functions include the MAJORITY function ($t = n/2$), the n-wise AND ($t = n$) and the n-wise OR ($t = 1$), the constant 0 ($t = n+1$) and the constant 1 ($t = 0$).

Let us denote by $s(n,d)$ the size of the smallest depth d AC^0 circuit computing MAJORITY. Recall the following bounds on $s(n,d)$:

- $s(n,d)$ has to be at least $2^{\Omega(n^{1/d-\epsilon})}$;
- $s(n,d)$ is at most $2^{n^{O(1/d)}}$.

Similarly, let us denote by $s_t(n,d)$ the size of the smallest depth d AC^0 circuit computing f_t. We have the following bounds:

- for any t, $s_t(n,d) = 2^{n^{O(1/d)}}$ by computing MAJORITY$(x1^{n-t+1}0^t)$;
- $s_1(n,1) = s_n(n,1) = 1$;
- for $t \ge 2$, $s_t(n,2) = O(n^t)$ by taking an OR of all $\binom{n}{t}$ t-wise ANDs of all sets of t bits.

Using pairwise independent hashing we can improve the above bound when t is polylogarithmic in n. We have the following theorem, due to Ragde and Wigderson.

Theorem 13. *Let $s_t(n,d)$ denote, as above, the size of the smallest depth d AC^0 circuit computing f_t. Then, for all $k \ge 1$ and $t = O(\log^k n)$, $s_t(n, \Theta(k)) = \mathtt{poly}(n)$, where the degree of the polynomial is independent of k.*

Proof. Our strategy will be to use pairwise independent hashing to reduce the problem to a problem instance whose size depends only on t. Specifically, we will construct an AC^0 circuit $C_1 : \{0,1\}^n \to \{0,1\}^{t^2+1}$, where we think of the output of the circuit as a pair $(y,b) = C_1(x)$. The essential property of C_1 is as follows:

$$C_1(x) = (y,b) \Rightarrow \begin{cases} \mathrm{wt}(y) = \mathrm{wt}(x) & b = 0 \\ \mathrm{wt}(x) > t & b = 1 \end{cases} \tag{1}$$

In other words, C_1 is either able to "decide" that $f_t(x) = 1$ or, in case of a failure, it reduces the problem to an instance of size t^2. To finish the construction of the circuit we need another depth d AC^0 circuit $C_2 : \{0,1\}^{t^2} \to \{0,1\}$ which computes $C_2(y) = f_t(y)$. As we discussed, we can bound the size of C_2 by $2^{t^{O(1/d)}}$. The final circuit C computes $\mathbf{1}[b=1] \vee (\mathbf{1}[C_1(x) = (y,0)] \wedge C_2(y))$. Let d_1, $d_2 = d$ be respectively the depths of C_1 and C_2, and s_1, s_2 – their sizes. Then the depth of C is $d_1 + d_2 + O(1)$, and the size of C is $s_1 + s_2 + O(1) = s_1 + 2^{t^{O(1/d)}}$. We will choose $d = \Omega(k)$ so that $s_2 = \mathtt{poly}(n)$. It remains to construct C_1 of polynomial size and constant depth.

To construct C_1 we will use pairwise independent hashing. Let $\mathcal{H} = \{h : [n] \to [t^2]\}$ be a pairwise independent hash family. By Lemma 9, for any $S \subseteq [n]$ of size at most t, there exists an $h \in \mathcal{H}$

such that $h|_S$ is one to one. We can take $S = \{i : x_i = 1\}$ and set b to be the indicator whether there exists an h which is one to one when restricted to S, and if such h exists also set $y_j = x_l$ if $h^{-1}(j) \cap S = \{l\}$ or 0 otherwise. Then C_1 would satisfy (1); however, we need to verify that this computation can be done with a constant depth AC^0 circuit.

To this end, define

$$\forall h \in \mathcal{H} : \beta_h = \neg \left(\bigvee_{i,j:h(i)=h(j)} x_i \wedge x_j \right).$$

Intuitively, β_h indicates whether h is "good", i.e. whether it is one to one when restricted to the indexes of the 1 bits in the input. Let us also define some canonical total ordering $\prec$ on $\mathcal{H}$; the following functions will indicate whether h is the first "good" h in $\mathcal{H}$:

$$\gamma_h = \beta_h \wedge \left(\bigwedge_{h' \prec h} \neg \beta_{h'} \right).$$

We are now ready to define b and y. First, b indicates whether all h are "bad", which, by Lemma 9 can only happen if $\mathrm{wt}(x) > t$.

$$b = \bigwedge_h \neg \beta_h$$

Finally, the bits y_j for $j \in [t^2]$ are defined with respect to the first "good" h.

$$y_j = \bigvee_h \left(\gamma_h \wedge \left(\bigvee_{i:h(i)=j} x_i \right) \right)$$

Each of γ_h, β_h, and, therefore, b and y, can be computed in constant depth. Also, computing each β_h requires $O(n^2)$ gates, each γ_h an additional $O(1)$ gates, b also an additional $O(1)$ gates, and, finally, each y_j – additional $O(|\mathcal{H}|)$ gates. The total size of C_1 is then $O(|\mathcal{H}|(n^2 + t^2)) = O(t^4 n^2 + t^6)$. □

Ragde and Wigderson also showed a construction with $o(n)$ gates.

7 Set Membership with 1 Bit Probe

We now turn our attention to the bit probe model. In this model, the data structure of Fredman, Komlos, and Szemeredi translates to space complexity of $O(n \log m)$ bits and query complexity of $O(\log m)$ bit probes. While the space complexity is optimal, we can no longer claim that the query complexity is optimal: our previous argument only implies that at least a single bit probe is necessary. In this section we explore whether a single bit probe is also sufficient while keeping the space complexity low. A deterministic single bit probe query algorithm is forced to probe a different bit for every query $i \in [m]$, and, therefore, the space complexity must be at least m bits. However, remarkably, we can still keep the space complexity low if we allow the query algorithm to be randomized.

In order to benefit from randomization, we are going to have to allow the query algorithm to make a mistake with some probability. We have two kinds of guarantees on such algorithms:

- *One-sided error*: a query algorithm for set membership with one-sided error ϵ answers "yes" on query i if $i \in S$ and answers "no" with probability greater than $1 - \epsilon$ if $i \notin S$;
- *Two-sided error*: a query algorithm for set membership with two-sided error ϵ gives the correct answer with probability greater than $1 - \epsilon$.

Notice that with both notions of error, the space lower bound of $\log \binom{m}{n}$ holds as long as $\epsilon < 1$ in the one-sided regime and $\epsilon < 1/2$ in the two-sided regime. This is because even a randomized query algorithm can still be used to recover exactly the input S, e.g. by enumerating over all possible random choices.

The beautiful data structures of Buhrman, Miltersen, Radhakrishnan and Venkatesh give nearly optimal space and bit probes in these settings. There exists a set membership data structure with space complexity $O(\frac{n^2}{\epsilon^2} \log m)$ bits and a single bit probe randomized one-sided error ϵ query algorithm. If we allow two-sided error ϵ, the space complexity can be brought down to $O(\frac{n}{\epsilon^2} \log m)$. Both results are optimal up to constant factors in the space complexity.

Theorem 14. *There exists a data structure of size $O(\frac{n^2}{\epsilon^2} \log m)$ and a randomized query algorithm that makes a single bit probe and has one-sided error ϵ.*

Proof. It is helpful to think of one-bit probe query algorithms in terms of a bipartite graph $G \subseteq [m] \times [l]$. For $v \in [m]$, let $\Gamma(v)$ be the neighborhood of v, and for $S \subseteq [m]$, let $\Gamma(S)$ be the neighborhood of S. Given input S, the data structure is the indicator vector of $\Gamma(S)$. Given a query $v \in [m]$, the query algorithm probes a uniform random neighbor u of v and answers "yes" if $u \in \Gamma(S)$. The space complexity of the data structure is l bits; the query algorithm has one-sided error ϵ if the following condition holds:

$$\forall S \subseteq [m](|S| = n), \forall v \notin S : |\Gamma(v) \cap \Gamma(S)| \leq \epsilon \Gamma(v). \tag{2}$$

We will choose G to be a-regular on the left and will require that

$$\forall v \neq w : |\Gamma(v) \cap \Gamma(w)| \leq b.$$

By an application of the probabilistic method, for every m, a and b, there exists such a G with $l = O((a^2/b)m^{1/b})$. If we set $b = \log m$ and $a = \frac{n \log m}{\epsilon}$, then we have $l = O(\frac{n^2}{\epsilon^2} \log m)$. For any v, $|\Gamma(v) = a|$, and, by the union bound,

$$|\Gamma(v) \cap \Gamma(S)| \leq \sum_{w \in S} |\Gamma(v) \cap \Gamma(w)| \leq nb.$$

Since we chose a and b so that $nb \leq \epsilon a$, we conclude that (2) holds. □