CS 880: Pseudorandomness and Derandomization 2/6/2013

Lecture 5: Applications of k-Wise Uniform Generators

Instructors: Holger Dell and Dieter van Melkebeek Scribe: Lubos Krcal

In this lecture we discuss two applications of k-wise uniform generators, namely (i) list decoding for the Hadamard code, which hinges on the linear structure of the simple pairwise uniform generator from the previous lecture, and (ii) hashing. Before embarking on those, we review a probability concentration result known as Chebyshev's inequality, which has a close connection to pairwise independence.

1 Chebyshev's Inequality

A good randomized algorithm should behave well not just in expectation, but with high probability. For example, if the outcome of the algorithm is a random variable whose expected value exceeds a desired threshold but the probability with which it exceeds that threshold is only 1%, then the algorithm is no good.

Concentration results in probability theory provide confidence levels with which a random variable X attains values around its expectation $\mathbb{E}[X]$. There are many such results for various types of random variables. Chebyshev's inequality is a concentration result that applies to every random variable X, and yields confidence levels as a function of its variance $\mathrm{Var}[X]$. Recall that

$$\mathrm{Var}[X] \doteq \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}[X^2] - \mathbb{E}[X]^2,$$

where the first equality is by definition, and the last equality follows from linearity of expectation. Note that since $\mathrm{Var}[X] \geq 0$ by definition, the last equality implies that $\mathbb{E}[X^2] \geq \mathbb{E}[X]^2$, which is the Cauchy-Schwarz inequality in disguise. For an indicator variable X, we have $\mathrm{Var}[X] = p \cdot (1-p)$, where p denotes the probability of the underlying event.

The following proposition bounds the probability that the deviation of a random variable from its expectation exceeds a certain threshold a as a function of the variance of the random variable.

Proposition 1 (Chebyshev's inequality). *For every random variable X and real $a > 0$,*

$$\Pr\Big[|X - \mathbb{E}(X)| \geq a\Big] \leq \frac{\mathrm{Var}[X]}{a^2}\,.$$

The proof of Chebyshev's inequality is an application of Markov's inequality, which is a concentration result that applies to every nonnegative random variable Y, and yields confidence levels as a function of its expected value.

Proposition 2 (Markov's inequality). *For every nonnegative random variable Y and real $b > 0$,*

$$\Pr\big[Y \geq b\big] \leq \frac{\mathbb{E}[Y]}{b}.$$

An instantiation of Markov's inequality states that the probability that a nonnegative random variable exceeds twice its expectation is at most 50%.

Proof (of Markov's inequality). Recall that $\mathbb{E}[Y] = \sum_y y \cdot \Pr[Y = y]$, where y ranges over all possible values of Y. We can split the sum into two sums with the given value b as the pivot:

$$\sum_y y \cdot \Pr[Y = y] = \sum_{y<b} y \cdot \Pr[Y = y] + \sum_{y \geq b} y \cdot \Pr[Y = y].$$

Since Y is nonnegative, the first sum (for $y < b$) is always nonnegative, whereas we can lower bound the second sum as follows:

$$\sum_{y \geq b} y \cdot \Pr[Y = y] \leq \sum_{y \geq b} b \cdot \Pr[Y = y] = b \cdot \sum_{y \geq b} \Pr[Y = y] = b \cdot \Pr[Y \geq b].$$

Combining the lower bounds for both sums yields

$$\mathbb{E}[Y] \geq b \cdot \Pr[Y \geq b]\,,$$

which is equivalent to the stated inequality. □

The proof of Chebyshev's inequality is now a direct invocation of Markov's inequality.

Proof (of Chebyshev's inequality). Apply Markov's inequality to the random variable $Y \doteq (X - \mathbb{E}[X])^2$, which is non-negative and has expectation $\mathbb{E}[Y] = \mathrm{Var}[X]$. For $b = a^2$ this shows that

$$\Pr\left[(X - \mathbb{E}[X])^2 \geq a^2\right] \leq \frac{\mathrm{Var}[X]}{a^2},$$

which is equivalent to the stated inequality. □

Connection with pairwise independence. Many concentration results apply to random variables X that are built up as the sum $X = \sum_{i=1}^{s} X_i$ of some components X_i that possess some degree of independence. The central limit theorem can be seen as a concentration result for a sequence of random variables X_i that are fully independent. Chebyshev's inequality applies to the random variable $X = \sum_{i=1}^{s} X_i$ no matter what the dependencies between the X_i's are, but yields particularly interesting bounds when the X_i's are pairwise independent. This is because the variance is additive when the X_i's are pairwise independent, as shown in the following proposition.

Proposition 3. *If the random variables $X_1, \ldots, X_s$ are pairwise independent, then*

$$\mathrm{Var}\left[\sum_{i=1}^{s} X_i\right] = \sum_{i=1}^{s} \mathrm{Var}[X_i]\,.$$

Proof.

$$
\begin{aligned}
\operatorname{Var}\left[\sum_{i=1}^{s} X_i\right] &= \mathbb{E}\left[\left(\sum_{i=1}^{s} X_i - \mathbb{E}\left[\sum_{i=1}^{s} X_i\right]\right)^2\right] && \text{(by the definition of variance)} \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{s} (X_i - \mathbb{E}[X_i])\right)^2\right] && \text{(by linearity of the inner expectation)} \\
&= \mathbb{E}\left[\sum_{i,j=1}^{s} (X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\right] && \text{(by spelling out the square)} \\
&= \sum_{i,j=1}^{s} \mathbb{E}\Big[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\Big] && \text{(by linearity of the outer expectation)} \\
&= \sum_{i=1}^{s} \mathbb{E}\Big[(X_i - \mathbb{E}[X_i])^2\Big] + \sum_{\substack{i,j\in[s] \\ i\neq j}} \mathbb{E}\Big[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\Big] && \text{(by splitting the sum)} \\
&= \sum_{i=1}^{s} \mathbb{E}\Big[(X_i - \mathbb{E}[X_i])^2\Big] + \sum_{\substack{i,j\in[s] \\ i\neq j}} \mathbb{E}\big[X_i - \mathbb{E}[X_i]\big] \cdot \mathbb{E}\big[X_j - \mathbb{E}[X_j]\big] && \text{(see below)} \\
&= \sum_{i=1}^{s} \mathbb{E}\Big[(X_i - \mathbb{E}[X_i])^2\Big] && \text{(see below)} \\
&= \sum_{i=1}^{s} \operatorname{Var}[X_i] && \text{(by definition of variance)}\,,
\end{aligned}
$$

where the third-to-last equality uses the fact that, for every $i \neq j$, the variables $Y_i \doteq X_i - \mathbb{E}[X_i]$ and $Y_j \doteq X_j - \mathbb{E}[X_j]$ are independent and therefore satisfy $\mathbb{E}[Y_i Y_j] = \mathbb{E}[Y_i]\mathbb{E}[Y_j]$, and the second-to-last equality follows because $\mathbb{E}[Y_i] = \mathbb{E}[X_i] - \mathbb{E}[X_i] = 0$. □

Note that the expected value is always additive: $\mathbb{E}[\sum_{i=1}^{s} X_i] = \sum_{i=1}^{s} \mathbb{E}[X_i]$ holds irrespective of the dependencies between the X_i's. In contrast, the variance is not always additive. For example, if all s variables X_i are identical, then $\operatorname{Var}[X] = \operatorname{Var}[sX_1] = s^2\operatorname{Var}[X_1]$, whereas additivity would require $\operatorname{Var}[X] = s\operatorname{Var}[X_1]$. But if the variables are pairwise independent, then the variance is additive.

Confidence boosting. We can reduce the probability of error of a randomized algorithm by running it s times and outputting the plurality answer, that is, the answer that occurred most frequently. Proposition 4 of Lecture 3 analyzed the technique when the s runs are fully independent. In that case, the error probability decreases exponentially with s. When the runs are only pairwise independent, we can use Chebyshev's inequality to show that the error decreases inversely proportional with s.

Proposition 4 (Confidence boosting). *Suppose that a randomized process outputs y with probability $\frac{1}{2} + \eta$. Then the plurality vote of s pairwise independent runs equals y with probability at least $1 - \delta$ for $s \geq \frac{1}{\delta\eta^2}$.*

Proof. Let X_i be the indicator that the ith run produces an answer other than y. Note that $\mathbb{E}[X_i] = \frac{1}{2} - \eta$ and $\mathrm{Var}[X_i] = \frac{1}{4} - \eta^2$. The only way the plurality vote can be different from y is when $X \doteq \sum_{i=1}^{s} X_i \geq \frac{s}{2}$, which implies that $|X - \mathbb{E}[X]| \geq \frac{s}{2} - s \cdot (\frac{1}{2} - \eta) = s \cdot \eta$. Since the runs are pairwise independent, Proposition 3 tells us that $\mathrm{Var}[X] = \sum_{i=1}^{s} \mathrm{Var}[X_i] = s \cdot (\frac{1}{4} - \eta^2)$. Thus, by Chebyshev's inequality, we can upper bound the probability that the plurality vote differs from y by

$$\Pr\Big[|X - \mathbb{E}[X]| \geq s\eta\Big] \leq \frac{s \cdot (\frac{1}{4} - \eta^2)}{(s\eta)^2} \leq \frac{1}{s\eta^2}\,,$$

which is at most δ if we set $s \geq 1/(\delta\eta^2)$. □

Let us compare the two ways we have seen for reducing the error from $\frac{1}{2} - \eta$ down to δ, namely taking the plurality output of a number of (i) fully independent runs and (ii) pairwise independent runs. We compare the number s of runs needed, as well as the number of truly random bits assuming a single run of the original algorithm uses r random bits. This number equals $r \cdot s$ in case (i). In case (ii), we can use the pairwise independent generator given by Theorem 5 from Lecture 4, which uses $2 \cdot \lceil \log_{2^r}(s) \rceil \cdot r = O(r + \log(s))$ truly random bits.

method	number of runs	number of random bits
(i) fully independent runs	$O(\log(\frac{1}{\delta}) \cdot \frac{1}{\eta^2})$	$O(r \cdot \log(\frac{1}{\delta}) \cdot \frac{1}{\eta^2})$
(ii) pairwise independent runs	$O(\frac{1}{\delta\eta^2})$	$O(r + \log(\frac{1}{\delta}) + \log(\frac{1}{\eta}))$

For fixed η, the required number of runs is exponentially smaller in $\frac{1}{\delta}$ for (i) than for (ii), but (ii) needs fewer random bits. Using expanders we will later see how to get the best of both worlds.

For fixed δ, the required number of runs for (i) and (ii) is the same up to constant factors, but (ii) only needs an additive number of extra random bits, whereas (i) needs a multiplicative extra. Using expanders we will later see how to do the same with no additional random bits and the same order of runs.

2 List Decoding the Hadamard Code

We are now ready to discuss our first application of pairwise independent generators. This is not a "standard" application, in which we would use a good pairwise independent pseudorandom generator for derandomization purposes. In fact, we'll be looking at a context in which full derandomization is impossible. Moreover, the argument will hinge on a special property of one of the pairwise independent PRGs we constructed, namely $\mathbb{F}_2$-linearity.

The context we are looking at is that of *list decoding* the Hadamard code. Recall that, as long as the number of errors is less than half the distance of a code, we can uniquely recover the correct encoding and thus the message. If the number of errors is somewhat larger, we can no longer guarantee that there is a unique codeword from which the received word can be obtained by introducing at most that many errors, but we can still hope that there are few such codewords. If so, we can try to list all of them, or all of the corresponding messages. The latter is referred to as list decoding.

Definition 1 (List decoding). *Let* $\mathrm{Enc} : \Sigma^k \to \Sigma^n$ *be a code. List decoding* Enc *up to relative error* ϵ *is the following problem: Given a received word* $z \in \Sigma^n$*, find all the messages* $x \in \Sigma^k$ *such that* $d_{\mathrm{H}}\big(\mathrm{Enc}(x), z\big) \leq \epsilon n$.

The Hadamard code Enc works over the binary alphabet $\Sigma = \{0,1\}$ and encodes $x \in \{0,1\}^k$ as a string of length $n = 2^k$, namely the inner product $\langle x, a\rangle$ of x with all strings $a \in \{0,1\}^k$. All codewords are at relative distance $\frac{1}{2}$ from each other, which means that we cannot uniquely decode if the fraction ϵ of errors is $\frac{1}{4}$ or more, and we cannot efficiently list decode if the fraction of errors is $\frac{1}{2}$ or more (since on input a valid codeword, the list would need to contain all messages of length k). In Lecture 3 we saw how to efficiently uniquely decode the Hadamard code for $\epsilon < \frac{1}{4}$. We now show how to efficiently list decode the Hadamard code for $\epsilon < \frac{1}{2}$. In order to do so, we model the received word $z \in \{0,1\}^{2^k}$ as a function $z : \{0,1\}^k \to \{0,1\}$, where the value of the function at $a \in \{0,1\}^k$ equals the bit $z(a) = z_a$ at position a of the string z.

Theorem 5. *There exists a randomized algorithm that, on input $\eta > 0$ and with oracle access to a function $z : \{0,1\}^k \to \{0,1\}$, produces a list that, with 99% certainty, contains all strings $x \in \{0,1\}^k$ that satisfy*

$$\Pr_{a \in_u \{0,1\}^k}\left[\langle x, a\rangle = z(a)\right] \geq \frac{1}{2} + \eta. \tag{1}$$

The algorithm runs in time polynomial in $\frac{k}{\eta}$.

A few remarks are in order. First, note that the algorithm is randomized and has oracle access to the received word z. Just like for the unique decoding procedure from Lecture 3, both are necessary in order to have any hope of achieving a running time that is polynomial in k.

Second, the algorithm only guarantees (with high probability) that the list *contains* all the solutions to the list decoding problem, but not that it *only consists of* solutions. Again, this is the best that we can hope for with a running time that is polynomial in k, for a similar reason: There may be messages x whose encoding agrees with the received word z in only slightly less than a fraction $\frac{1}{2} + \eta$ of the positions. Such an x should be included in the list for a slightly modified received word $z \in \{0,1\}^{2^k}$, and there is no way for an algorithm that runs in time polynomial in k – even a randomized one – to detect this difference with reasonable confidence. What we *can* efficiently guarantee is that no x on the list has an encoding that agrees with z in significantly less than $\frac{1}{2} + \eta$ of the positions.

Finally, the existence of the algorithm implies that the number of solutions to the list decoding problem is polynomial in $O(\frac{k}{\eta^2})$, which is a nontrivial result by itself. In fact, the following lemma, which captures the crux of the proof of Theorem 5, implies an upper bound of $O(\frac{k}{\eta^2})$ on the number of solutions. In a later lecture we will see how to improve this upper bound to $\frac{1}{4\eta^2}$, which is tight.

Lemma 6. *There exists a randomized algorithm that, on input $\eta > 0$ and oracle access to a function $z : \{0,1\}^k \to \{0,1\}$, produces a list of size $O(\frac{k}{\eta^2})$ such that, for every string $x \in \{0,1\}^k$ satisfying (1), the probability that x is included in the list is at least 50%. The algorithm runs in time polynomial in $\frac{k}{\eta}$.*

Corollary 7. *For every $\eta > 0$ and function $z : \{0,1\}^k \to \{0,1\}$, the number of $x \in \{0,1\}^k$ satisfying 1 is $O(\frac{k}{\eta^2})$.*

Proof (of Corollary 7). Let S denote the set of solutions and L the list that the algorithm of Lemma 6 outputs. The statement of the lemma implies that

$$|S| \cdot \frac{1}{2} \leq \sum_{x \in S} \Pr[x \in L] \leq \sum_x \Pr[x \in L] = \sum_x \mathbb{E}[I[x \in L]] = \mathbb{E}[|L|] = O(\frac{k}{\eta^2}),$$

which implies that $|S| = O(\frac{k}{\eta^2})$. □

The proof of Theorem 5 is immediate from Lemma 6 and Corollary 7.

Proof (of Theorem 5). Let S denote the set of solutions. We run the algorithm of Lemma 6 a number t times and output the concatenation of the t lists. This yields a list L such that, for every fixed $x \in S$, $\Pr[x \notin L] \leq \frac{1}{2^t}$. Hence, by the union bound, $\Pr[(\exists x \in S)\, x \notin L] \leq |S|/2^t$, which by the upper bound on $|S|$ given by Corollary 7 is at most 1% for $t \geq \log(\frac{k}{\eta^2}) + O(1)$. □

We now get to the crux of the argument, which is Lemma 6. Let us fix a solution x, i.e., a string $x \in \{0,1\}^k$ satisfying (1). For starters, let us recall the basis for the local decoding procedure from Lecture 3: In order to find the ith bit of x, pick $a \in \{0,1\}^k$ uniformly at random and output $z(a) + z(a + e_i)$. We argued that

$$\Pr_{a \in_u \{0,1\}^k}[z(a) + z(a + e_i) \neq x_i] \leq 1 - 2\eta. \tag{2}$$

The reason was that the predicate underlying (2) holds if z agrees with the encoding of x at the two positions a and $a + e_i$, and by the union bound this fails with probability at most $(\frac{1}{2} - \eta) + (\frac{1}{2} - \eta)$. For $\eta > \frac{1}{4}$, (2) shows that the probability of error remains bounded below $\frac{1}{2}$, after which we can apply the usual confidence boosting techniques to obtain the correct result with high probability. When $\eta \leq \frac{1}{4}$, however, the upper bound in (2) becomes meaningless.

In order to get around this issue, consider the following "procedure" for finding the ith bit of x: Pick $a \in \{0,1\}^k$ uniformly at random, and output $\langle x, a\rangle + z(a + e_i)$. We write "procedure" between quotes because it isn't clear how we can find the value $\langle x, a\rangle$ of the correct encoding of x at a, but let us assume for now that we can. Since

$$\Pr_{a \in_u \{0,1\}^k}[\langle x, a\rangle + z(a + e_i) \neq x_i] \leq \frac{1}{2} - \eta,$$

we are in a position to apply confidence boosting for any $\eta > 0$. In particular, if we take the majority vote for t choices of a produced by a pairwise uniform generator $G : \Sigma^\ell \to \Sigma^t$ for $\Sigma = \{0,1\}^k$, Proposition 4 guarantees us that

$$\Pr_{\sigma \in_u \Sigma^\ell}\Big[\mathrm{maj}_a(\langle x, a\rangle + z(a + e_i)) \neq x_i\Big] \leq \frac{1}{t\eta^2},$$

where the majority is taken over all the t components a of the random variable $G(\sigma)$. By the union bound, we get

$$\Pr_{\sigma \in_u \Sigma^\ell}\Big[\big(\mathrm{maj}_a(\langle x, a\rangle + z(a + e_i))\big)_{i=1}^k \neq x\Big] \leq \frac{k}{t\eta^2}.$$

Thus, for any $t \geq \frac{2k}{\eta^2}$, the probability that our "procedure" outputs x is at least 50%.

How do we obtain the values $\langle x, a\rangle$ we need? Consider the simple pairwise independent generator that generalizes Lemma **??** from Lecture 4: $G(\sigma) = (\sum_{j=1}^\ell c_j\sigma_j)_c$, where the arithmetic is over $\mathbb{F}_2^k$ and c ranges over $\{0,1\}^\ell \setminus \{0^\ell\}$, so $t = 2^\ell - 1$. Note that for $a_c \doteq \sum_{j=1}^\ell c_j\sigma_j$,

$$\langle x, a_c\rangle = \sum_{j=1}^{\ell} c_j\langle x, \sigma_j\rangle, \tag{3}$$

which means that in order to know all $2^\ell - 1$ values $\langle x, a_c \rangle$ it suffices to know the ℓ values $\langle x, \sigma_j \rangle$ for $j \in [\ell]$. We do not really know those values, but since there are so few of them, we can just try all possibilities. That is, for a given seed σ, we range over all 2^ℓ possible values for $(\langle x, \sigma_j \rangle)_{j=1}^{\ell}$, use those values to compute $\langle x, a_c \rangle$ via (3) for all $c \in \{0,1\}^\ell \setminus \{0^\ell\}$, and include $(\mathrm{maj}_c(\langle x, a_c \rangle + z(a_c + e_i)))_{i=1}^{k}$ in our list of candidate solutions. By the above analysis, the list contains x with probability at least 50% if we set $t \geq \frac{2k}{\eta^2}$. The size of the list equals $2^\ell = t + 1$. If we choose the smallest $t \geq \frac{2k}{\eta^2}$ of the form $t = 2^\ell - 1$, the list size is $O(\frac{k}{\eta^2})$. The running time of the resulting procedure is polynomial in k and $\frac{1}{\eta}$. This finishes the proof of Lemma 6.

Note that the resulting algorithm uses a lot of randomness, namely $k \cdot \ell = O(k \log(\frac{k}{\eta}))$ truly random bits. Indeed, the reason we use the (particular) pairwise uniform PRG is not to save on random bits, but because every pseudorandom sequence can be written as a $\mathbb{F}_2$-linear combination of a number of sequences that is logarithmic in the length of the sequence. This allows us to produce a polynomial number of pairwise independent positions a which we can use for boosting à la Chebyshev and for which we can generate in polynomial time all possible restrictions of a valid codeword to the subset of those positions a.

3 Hashing

Hashing is a technique to deal with the dictionary problem, in which we want to store n (key,value) pairs where the keys come from a universe $[u]$ that is much larger than n. A typical setting is a compiler keeping track of the variables that are in use; in this case the keys are the variable names. In the static version of the problem, the set S of stored keys is fixed and we only allow look-ups, i.e., retrieving the value corresponding to a given key, or reporting that there is no value stored for the given key. In the dynamic version, the set S can change over time via additions and deletions of (key,value) pairs.

One possible solution is to create a table with one entry for each possible key value in $[u]$, and use the key value as the index into that table. This allows us to perform look-ups in constant time, but requires an exorbitant amount of memory space and set-up time because $u \gg n$. Another possibility is to store S in a balanced search tree, in which case we only need about the memory space to write down the n (key,value) pairs but the operations now take $\Theta(\log n)$ time. Hashing is an attempt to combine the best of both worlds. It uses a table of size $m \approx n$, and stores a (key,value) pair (x, v) in slot $h(x)$ of the table. The name of the game is to find a mapping $h : [u] \to [m]$, called the hash function, that is simple to compute and avoids collisions, i.e., no two distinct keys $x, y \in S$ map to the same slot $h(x) = h(y)$. This results in constant look-up time and only requires memory comparable to that of a balanced search tree.

Having no collisions at all may be a lot to ask, but finding a hash function that results in few collisions is easy, at least in a randomized setting – a random choice will do. More precisely, if we pick h uniformly from the class $\mathcal{H}$ of all functions from $[u]$ to $[m]$, then the expected number of collisions with any given key $x \in S$ is no more than the load factor $\alpha \doteq n/m$ of the table. This is because

$$E_{h \in_u \mathcal{H}}[|\{y \in S \text{ s.t. } h(x) = h(y)\}|] = \sum_{x \neq y \in S} \Pr_{h \in_u \mathcal{H}}[h(x) = h(y)] = \frac{n-1}{m} \leq \alpha. \tag{4}$$

However, a random hash function h is not simple – it takes $2^u \cdot \log m$ bits to specify. Instead, we'd

like to use functions from a much smaller family $\mathcal{H}$ of hash functions that can be described and evaluated efficiently.

Note that the only property of $\mathcal{H}$ we need for (4) to hold is that

$$(\forall x, y \in [u] \text{ with } x \neq y) \Pr_{h \in_u \mathcal{H}}[h(x) = h(y)] = \frac{1}{m}. \tag{5}$$

A family $\mathcal{H}$ with property (5) is called *universal*, and it suffices that the marginal distribution of the values of h on any two distinct inputs $x, y \in [u]$ is uniform. A family with the latter property is called 2-uniform. More generally, we have the following definition for any positive integer k.

Definition 2. *A family $\mathcal{H}$ of functions $h : [u] \to [m]$ is k-uniform if for every k distinct elements $x_1, x_2, \ldots, x_k \in [u]$, the distribution of $(h(x_i))_{i=1}^k$ is uniform over $[m]^k$ when h is chosen uniformly from $\mathcal{H}$.*

There is a close connection between k-uniform families of hash functions $\mathcal{H}$ and k-wise uniform generators $G : \{0,1\}^\ell \to [m]$, where the seed of G is used to describe a hash function, and the value of the hash function at $x \in [u]$ is the xth component of $G(\sigma)$, i.e., $h_\sigma(x) = (G(\sigma))_x$. Using the k-wise uniform generators given by Theorem 5 from Lecture 4, we obtain seed length $\ell = O(k \cdot \log u)$, so the corresponding k-uniform family $\mathcal{H}$ of hash functions has size $|\mathcal{H}| = u^{O(k)}$. For example, when m is a power of 2 and u is a power of m, the following family is 2-uniform: $\mathcal{H} = \{h_{a,b} \mid a.b \in \mathbb{F}_u\}$, where $h_{a,b}(x)$ is the last $\log m$ bits of $ax + b$ when we interpret $x \in [u]$ as an element of $\mathbb{F}_u$, and the arithmetic is carried out over that field. Each function from this family is described by $2\log(u)$ bits. For $m \ll u$ we can save almost a factor of 2 by using Exercise 1 of the previous lecture. It yields a 2-uniform family where m and u are powers of 2, and each function from the family is described by $\log(u) + 2\log(m)$ bits.

In lecture 1 we used a family of hash functions $h_p(x) = x \bmod p$, where p is a prime consisting of $c \log\log(u)$ bits, and therefore $m = \log(u)^c$. The resulting family is not universal stricto sensu because the probability that two distinct elements collide is more than $\frac{1}{m}$. However, the collision probability is only slightly more than $\frac{1}{m}$, and therefore the family pretty much shares the properties of universal families.

When we use a universal family, the expected number of collisions for any $x \in S$ is less than $\alpha \doteq \frac{n}{m}$. Even when we keep α under control, some collisions will occur and we need a way to deal with them. Various strategies have been considered, including the following.

- Chaining.
  The keys in S that hash to the same slot in $[m]$ are concatenated in a linked list, to which the entry in the hash table points. For the class $\mathcal{H}$ of all hash functions, (4) implies that the expected look-up time is bounded by $1 + \alpha$, and our analysis shows that the same holds for any universal family $\mathcal{H}$.

- Linear probing.
  This is a particular method of open addressing, which means that all keys are stored in the hash table itself. Specifically, when we want to place a key x into a slot of the hash table that is already in use, we search the successive slots in the hash table until we find an empty one, and store x there. It was known for a long time that for the class $\mathcal{H}$ of all hash functions, the expected look-up time is $O(\frac{1}{(1-\alpha)^2})$. A few years ago it was shown that the same holds for every 5-uniform family $\mathcal{H}$ (but not for every 4-uniform one) [PPR11].

Note that we only made claims about the expected look-up times, whereas we really want good behavior with very high probability. In addition, there are many practical considerations that play a role in the choice of the hash functions. For example, linear probing has good cache behavior because of the fact that it accesses successive cells, which may counterbalance its worse expected look-up time compared to chaining.

References

[PPR11] Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with 5-wise independence. *SIAM Review*, 53(3):547–558, 2011.