

**DigiSem**

Wir beschaffen und  
digitalisieren

*u<sup>b</sup>*

---

<sup>b</sup>  
**UNIVERSITÄT  
BERN**

Universitätsbibliothek Bern

Informationen Digitale Semesterapparate:

[www.digisem.unibe.ch](http://www.digisem.unibe.ch)

Fragen und Support:

[digisem@ub.unibe.ch](mailto:digisem@ub.unibe.ch) oder Telefon 031 631 93 26

**Probability and Computing**  
**Randomization and Probabilistic**  
**Techniques in Algorithms and**  
**Data Analysis**

**Second Edition**

**Michael Mitzenmacher      Eli Upfal**



**CAMBRIDGE**  
UNIVERSITY PRESS

**CAMBRIDGE**  
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

4843/24, 2nd Floor, Ansari Road, Daryaganj, Delhi - 110002, India

79 Anson Road, #06-04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

[www.cambridge.org](http://www.cambridge.org)

Information on this title: [www.cambridge.org/9781107154889](http://www.cambridge.org/9781107154889)

10.1017/9781316651124

© Michael Mitzenmacher and Eli Upfal 2017

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2017

Printed in the United Kingdom by Clays, St Ives plc

*A catalogue record for this publication is available from the British Library.*

*Library of Congress Cataloging in Publication Data*

Names: Mitzenmacher, Michael, 1969– author. | Upfal, Eli, 1954– author.

Title: Probability and computing / Michael Mitzenmacher Eli Upfal.

Description: Second edition. | Cambridge, United Kingdom ;

New York, NY, USA : Cambridge University Press, [2017] |

Includes bibliographical references and index.

Identifiers: LCCN 2016041654 | ISBN 9781107154889

Subjects: LCSH: Algorithms. | Probabilities. | Stochastic analysis.

Classification: LCC QA274.M574 2017 | DDC 518/.1 – dc23

LC record available at <https://lccn.loc.gov/2016041654>

ISBN 978-1-107-15488-9 Hardback

Additional resources for this publication at [www.cambridge.org/Mitzenmacher](http://www.cambridge.org/Mitzenmacher).

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet Web sites referred to in this publication and does not guarantee that any content on such Web sites is, or will remain, accurate or appropriate.

## CHAPTER FIFTEEN

# Pairwise Independence and Universal Hash Functions

---

In this chapter we introduce and apply a limited notion of independence, known as  $k$ -wise independence, focusing in particular on the important case of pairwise independence. Applying limited dependence can allow us to reduce the amount of randomness used by a randomized algorithm, in some cases enabling us to convert a randomized algorithm into an efficient deterministic one. Limited dependence is also used in the design of universal and strongly universal families of hash functions, giving space- and time-efficient data structures. We consider why universal hash functions are effective in practice and show how they lead to simple perfect hash schemes. Finally, we apply these ideas to the design of effective and practical approximation algorithms for finding frequent objects in data streams, generalizing the Bloom filter data structure introduced in Chapter 5.

### 15.1. Pairwise Independence

Recall that in Chapter 2 we defined a set of events  $E_1, E_2, \dots, E_n$  to be mutually independent if, for any subset  $I \subseteq [1, n]$ ,

$$\Pr\left(\bigcap_{i \in I} E_i\right) = \prod_{i \in I} \Pr(E_i).$$

Similarly, we defined a set of random variables  $X_1, X_2, \dots, X_n$  to be mutually independent if, for any subset  $I \subseteq [1, n]$  and any values  $x_i, i \in I$ ,

$$\Pr\left(\bigcap_{i \in I} (X_i = x_i)\right) = \prod_{i \in I} \Pr(X_i = x_i).$$

Mutual independence is often too much to ask for. Here, we examine a more limited notion of independence that proves useful in many contexts:  $k$ -wise independence.

**Definition 15.1:**

1. A set of events  $E_1, E_2, \dots, E_n$  is  $k$ -wise independent if, for any subset  $I \subseteq [1, n]$  with  $|I| \leq k$ ,

$$\Pr\left(\bigcap_{i \in I} E_i\right) = \prod_{i \in I} \Pr(E_i).$$

2. A set of random variables  $X_1, X_2, \dots, X_n$  is  $k$ -wise independent if, for any subset  $I \subseteq [1, n]$  with  $|I| \leq k$  and for any values  $x_i, i \in I$ ,

$$\Pr\left(\bigcap_{i \in I} (X_i = x_i)\right) = \prod_{i \in I} \Pr(X_i = x_i).$$

3. The random variables  $X_1, X_2, \dots, X_n$  are said to be pairwise independent if they are 2-wise independent. That is, for any pair  $i, j$  and any values  $a, b$ ,

$$\Pr((X_i = a) \cap (X_j = b)) = \Pr(X_i = a) \Pr(X_j = b).$$

**15.1.1. Example: A Construction of Pairwise Independent Bits**

A random bit is uniform if it assumes the values 0 and 1 with equal probability. Here we show how to derive  $m = 2^b - 1$  uniform pairwise independent bits from  $b$  independent, uniform random bits  $X_1, \dots, X_b$ .

Enumerate the  $2^b - 1$  nonempty subsets of  $\{1, 2, \dots, b\}$  in some order, and let  $S_j$  be the  $j$ th subset in this ordering. Set

$$Y_j = \bigoplus_{i \in S_j} X_i,$$

where  $\bigoplus$  is the exclusive-or operation. Equivalently, we could write this as

$$Y_j = \sum_{i \in S_j} X_i \bmod 2.$$

**Lemma 15.1:** *The  $Y_j$  are pairwise independent uniform bits.*

**Proof:** We first show that, for any nonempty set  $S_j$ , the random bit

$$Y_j = \bigoplus_{i \in S_j} X_i$$

is uniform. This follows easily using the principle of deferred decisions (see Section 1.3). Let  $z$  be the largest element of  $S$ . Then

$$Y_j = \left( \bigoplus_{i \in S_j - \{z\}} X_i \right) \oplus X_z.$$

Suppose we reveal the values for  $X_i$  for all  $i \in S_j - \{z\}$ . Then it is clear that the value of  $X_z$  determines the value of  $Y_j$  and that  $Y_j$  will take on the values 0 and 1 with equal probability.

Now consider any two variables  $Y_k$  and  $Y_\ell$  with their corresponding sets  $S_k$  and  $S_\ell$ . Without loss of generality, let  $z$  be an element of  $S_\ell$  that is not in  $S_k$  and consider, for any values  $c, d \in \{0, 1\}$ ,

$$\Pr(Y_\ell = d \mid Y_k = c).$$

We claim, again by the principle of deferred decisions, that this probability is  $1/2$ . For suppose that we reveal the values for  $X_i$  for all  $i$  in  $(S_k \cup S_\ell) - \{z\}$ . Even though this determines the value of  $Y_k$ , the value of  $X_z$  will determine  $Y_\ell$ . The conditioning on the value of  $Y_k$  therefore does not change that  $Y_\ell$  is equally likely to be 0 or 1. Hence

$$\begin{aligned} \Pr((Y_k = c) \cap (Y_\ell = d)) &= \Pr(Y_\ell = d \mid Y_k = c) \Pr(Y_k = c) \\ &= 1/4. \end{aligned}$$

Since this holds for any values of  $c, d \in \{0, 1\}$ , we have proven pairwise independence. ■

### 15.1.2. Application: Derandomizing an Algorithm for Large Cuts

In Chapter 6, we examined a simple randomized algorithm for finding a large cut in an undirected graph  $G = (V, E)$ : the algorithm places each vertex on one side of the cut independently with probability  $1/2$ . The expected value of a cut generated this way is  $m/2$ , where  $m$  is the number of edges in the graph. We also showed (in Section 6.3) that this algorithm could be derandomized effectively using conditional expectations.

Here we present another way to derandomize this algorithm, using pairwise independence. This argument exemplifies the approach of derandomization using  $k$ -wise independence.

Suppose that we have a collection  $Y_1, Y_2, \dots, Y_n$  of pairwise independent bits, where  $n = |V|$  is the number of vertices in the graph. We define our cut by putting all vertices  $i$  with  $Y_i = 0$  on one side of the cut and all vertices  $i$  with  $Y_i = 1$  on the other side of the cut. We show that, in this case, the expected number of edges that crosses the cut remains  $m/2$ . That is, we do not require complete independence to analyze the expectation; pairwise independence suffices.

Recall the argument of Section 6.2.1: number the edges from 1 to  $m$ , and let  $Z_i = 1$  if the  $i$ th edge crosses the cut and  $Z_i = 0$  otherwise. Then  $Z = \sum_{i=1}^m Z_i$  is the number of edges crossing the cut, and

$$\mathbf{E}[Z] = \mathbf{E}\left[\sum_{i=1}^m Z_i\right] = \sum_{i=1}^m \mathbf{E}[Z_i].$$

Let  $a$  and  $b$  be the two vertices adjacent to the  $i$ th edge. Then

$$\Pr(Z_i = 1) = \Pr(Y_a \neq Y_b) = 1/2,$$

where we have used the pairwise independence of  $Y_a$  and  $Y_b$ . Hence  $E[Z_i] = 1/2$ , and it follows that  $E[Z] = m/2$ .

Now let our  $n$  pairwise independent bits  $Y_1, \dots, Y_n$  be generated from  $b$  independent, uniform random bits  $X_1, X_2, \dots, X_b$  in the manner of Lemma 15.1 (here  $b = \lceil \log_2(n+1) \rceil$ ). Then  $E[Z] = m/2$  for the resulting cut, where the sample space is just all the possible choices for the initial  $b$  random bits. By the probabilistic method (specifically, Lemma 6.2), there is some setting of the  $b$  bits that gives a cut with value at least  $m/2$ . We can try all possible  $2^b$  settings for the bits to find such a cut. Since  $2^b$  is  $O(n)$  and since, for each cut, the number of crossing edges can easily be calculated in  $O(m)$  time, it follows that we can find a cut with at least  $m/2$  crossing edges deterministically in  $O(mn)$  time.

Although this approach does not appear to be as efficient as the derandomization of Section 6.3, one redeeming feature of the scheme is that it is trivial to parallelize. If we have sufficiently many processors available, then each of the  $\Omega(n)$  possibilities for the random bits  $X_1, X_2, \dots, X_b$  can be assigned to a single processor, with each possibility giving a cut. The parallelization reduces the running time by a factor of  $\Omega(n)$  using  $O(n)$  processors. In fact, using  $O(mn)$  processors, we can assign a processor for each combination of a specific edge with a specific sequence of random bits and then determine, in constant time, whether the edge crosses the cut for that setting of the random bits. After that, only  $O(\log n)$  time is necessary to collect the results and find the large cut.

### 15.1.3. Example: Constructing Pairwise Independent Values Modulo a Prime

We consider another construction that provides pairwise independent values  $Y_0, Y_1, \dots, Y_{p-1}$  that are uniform over the values  $\{0, 1, \dots, p-1\}$  for a prime  $p$ . Our construction requires only two independent, uniform values  $X_1$  and  $X_2$  over  $\{0, 1, \dots, p-1\}$ , from which we derive

$$Y_i = X_1 + iX_2 \bmod p \quad \text{for } i = 0, \dots, p-1.$$

**Lemma 15.2:** *The variables  $Y_0, Y_1, \dots, Y_{p-1}$  are pairwise independent uniform random variables over  $\{0, 1, \dots, p-1\}$ .*

**Proof:** It is clear that each  $Y_i$  is uniform over  $\{0, 1, \dots, p-1\}$ , again by applying the principle of deferred decisions. Given  $X_2$ , the  $p$  distinct possible values for  $X_1$  give  $p$  distinct possible values for  $Y_i$  modulo  $p$ , each of which is equally likely.

Now consider any two variables  $Y_i$  and  $Y_j$ . We wish to show that, for any  $a, b \in \{0, 1, \dots, p-1\}$ ,

$$\Pr((Y_i = a) \cap (Y_j = b)) = \frac{1}{p^2},$$

which implies pairwise independence. The event  $Y_i = a$  and  $Y_j = b$  is equivalent to

$$X_1 + iX_2 = a \bmod p \quad \text{and} \quad X_1 + jX_2 = b \bmod p.$$

This is a system of two equations and two unknowns with just one solution:

$$X_2 = \frac{b-a}{j-i} \bmod p \quad \text{and} \quad X_1 = a - \frac{i(b-a)}{j-i} \bmod p.$$

Since  $X_1$  and  $X_2$  are independent and uniform over  $\{0, 1, \dots, p-1\}$ , the result follows. ■

This proof can be extended to the following useful result: given  $2n$  independent, uniform random bits, one can construct up to  $2^n$  pairwise independent and uniform strings of  $n$  bits. The extension requires knowledge of finite fields, so we only sketch the result here. The setup and proof are exactly the same as for Lemma 15.2 except that, instead of working modulo  $p$ , we perform all arithmetic in a fixed finite field with  $2^n$  elements (such as the field  $GF(2^n)$  of all polynomials with coefficients in  $GF(2)$  modulo some irreducible polynomial of degree  $n$ ). That is, we assume a fixed one-to-one mapping  $f$  from strings of  $n$  bits, which can also be thought of as numbers in  $\{0, 1, \dots, 2^n - 1\}$ , to field elements. We let

$$Y_i = f^{-1}(f(X_1) + f(i) \cdot f(X_2)),$$

where  $X_1$  and  $X_2$  are chosen independently and uniformly over  $\{0, 1, \dots, 2^n - 1\}$ ,  $i$  runs over the values  $\{0, 1, \dots, 2^n - 1\}$ , and the addition and multiplication are performed over the field. The  $Y_i$  are then pairwise independent.

## 15.2. Chebyshev's Inequality for Pairwise Independent Variables

Pairwise independence is much weaker than mutual independence. For example, we can use Chernoff bounds to evaluate the tail distribution of a sum of independent random variables, but we cannot directly apply a Chernoff bound if the  $X_i$  are only pairwise independent. However, pairwise independence is strong enough to allow for easy calculation of the variance of the sum, which allows for a useful application of Chebyshev's inequality.

**Theorem 15.3:** *Let  $X = \sum_{i=1}^n X_i$ , where the  $X_i$  are pairwise independent random variables. Then*

$$\text{Var}[X] = \sum_{i=1}^n \text{Var}[X_i].$$

**Proof:** We saw in Chapter 3 that

$$\text{Var} \left[ \sum_{i=1}^n X_i \right] = \sum_{i=1}^n \text{Var}[X_i] + 2 \sum_{i < j} \text{Cov}(X_i, X_j),$$

where

$$\text{Cov}(X_i, X_j) = \mathbf{E}[(X_i - \mathbf{E}[X_i])(X_j - \mathbf{E}[X_j])] = \mathbf{E}[X_i X_j] - \mathbf{E}[X_i] \mathbf{E}[X_j].$$



Since  $X_1, X_2, \dots, X_n$  are pairwise independent, it is clear (by the same argument as in Theorem 3.3) that for any  $i \neq j$  we have

$$\mathbf{E}[X_i X_j] - \mathbf{E}[X_i] \mathbf{E}[X_j] = 0.$$

Therefore,

$$\mathbf{Var}[X] = \sum_{i=1}^n \mathbf{Var}[X_i].$$

Applying Chebyshev's inequality to the sum of pairwise independent variables yields the following.

**Corollary 15.4:** *Let  $X = \sum_{i=1}^n X_i$ , where the  $X_i$  are pairwise independent random variables. Then*

$$\Pr(|X - \mathbf{E}[X]| \geq a) \leq \frac{\mathbf{Var}[X]}{a^2} = \frac{\sum_{i=1}^n \mathbf{Var}[X_i]}{a^2}.$$

### 15.2.1. Application: Sampling Using Fewer Random Bits

We apply Chebyshev's inequality for pairwise independent random variables to obtain a good approximation through sampling. This uses less randomness than the natural approach based on Chernoff bounds.

Suppose that we have a function  $f : \{0, 1\}^n \rightarrow [0, 1]$  mapping  $n$ -bit vectors into real numbers. Let  $\bar{f} = (\sum_{x \in \{0, 1\}^n} f(x)) / 2^n$  be the average value of  $f$ . We want to compute a  $1 - \delta$  confidence interval for  $\bar{f}$ . That is, we wish to find an interval  $[\bar{f} - \varepsilon, \bar{f} + \varepsilon]$  such that

$$\Pr(\bar{f} \in [\bar{f} - \varepsilon, \bar{f} + \varepsilon]) \geq 1 - \delta.$$

As a concrete example, suppose that we have an integrable function  $g : [0, 1] \rightarrow [0, 1]$  and that the derivative of  $g$  exists with  $|g'(x)| \leq C$  for some fixed constant  $C$  over the entire interval  $(0, 1)$ . We are interested in  $\int_{x=0}^1 g(x) dx$ . There may be no direct way to compute this integral exactly, but through sampling we can obtain a good estimate. If  $X$  is a uniform random variable on  $[0, 1]$ , then  $\mathbf{E}[g(X)] = \int_{x=0}^1 g(x) dx$  by the definition of the expectation of a continuous random variable. By taking the average of multiple independent samples, we can approximate the integral. If our source of randomness generates only random bits instead of random real numbers, then we might approximate the integral as follows. For a string of bits  $x \in \{0, 1\}^n$ , we may interpret  $x$  as a real number  $\bar{x} \in [0, 1]$  by considering it as a decimal in binary; for example, 11001 would correspond to  $0.11001 = 25/32$ . Let  $f(x)$  denote the value of the function  $g$  at the decimal value  $\bar{x}$ . Then, for any integer  $i$  with  $0 \leq i \leq 2^n - 1$ , for  $y \in [i/2^n, (i+1)/2^n)$  we have

$$f\left(\frac{i}{2^n}\right) - \frac{C}{2^n} \leq g(y) \leq f\left(\frac{i}{2^n}\right) + \frac{C}{2^n}.$$

It follows that

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left( f(x) - \frac{C}{2^n} \right) \leq \int_{x=0}^1 g(x) dx \leq \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \left( f(x) + \frac{C}{2^n} \right).$$

By taking  $n$  sufficiently large, we can guarantee that  $\bar{f} = (\sum_{x \in \{0,1\}^n} f(x))/2^n$  differs from the integral of  $g$  by at most a constant  $\gamma$ . In this case, a confidence interval  $[\bar{f} - \varepsilon, \bar{f} + \varepsilon]$  for  $\bar{f}$  would yield a confidence interval  $[\bar{g} - \varepsilon - \gamma, \bar{g} + \varepsilon + \gamma]$  for the integral of  $g$ .

We could handle the problem of finding a confidence interval for the average value  $\bar{f}$  by using independent samples and applying a Chernoff bound. That is, suppose that we sample uniformly with replacement random points in  $\{0,1\}^n$ , evaluate  $f$  at all of these points, and take the average of our samples. This is similar to the parameter estimation of Section 4.2.3. Theorem 15.5 is an immediate consequence of the following Chernoff bound, which can be derived using Exercises 4.13 and 4.19. If  $Z_1, Z_2, \dots, Z_m$  are independent, identically distributed, real-valued random variables with mean  $\mu$  that take on one of a finite possible set of values in the range  $[0, 1]$ , then

$$\Pr \left( \left| \sum_{i=1}^m Z_i - m\mu \right| \geq \varepsilon m \right) \leq 2e^{-2m\varepsilon^2}.$$

**Theorem 15.5:** Let  $f : \{0,1\}^n \rightarrow [0,1]$  and  $\bar{f} = (\sum_{x \in \{0,1\}^n} f(x))/2^n$ . Let  $X_1, \dots, X_m$  be chosen independently and uniformly at random from  $\{0,1\}^n$ . If  $m > \ln(2/\delta)/2\varepsilon^2$ , then

$$\Pr \left( \left| \frac{1}{m} \sum_{i=1}^m f(X_i) - \bar{f} \right| \geq \varepsilon \right) \leq \delta.$$

Although the exact choice of  $m$  depends on the Chernoff bound used, in general this straightforward approach requires  $\Omega(\ln(1/\delta)/\varepsilon^2)$  samples to achieve the desired bounds.

A possible problem with this approach is that it requires a large number of random bits to be available. Each sample of  $f$  requires  $n$  independent bits, so applying Theorem 15.5 means that we need at least  $\Omega(n \ln(1/\delta)/\varepsilon^2)$  independent, uniform random bits to obtain an approximation that has additive error at most  $\varepsilon$  with probability at least  $1 - \delta$ .

A related problem arises when we need to record how the samples were obtained, so that the work can be reproduced and verified at a later time. In this case, we also need to store the random bits used for archival purposes. In this case, using fewer random bits would lessen the storage requirements.

We can use pairwise independent samples to obtain a similar approximation using less randomness. Let  $X_1, \dots, X_m$  be pairwise independent points chosen from  $\{0,1\}^n$ , and let  $Y = (\sum_{i=1}^m f(X_i))/m$ . Then  $\mathbb{E}[Y] = \bar{f}$ , and we can apply Chebyshev's

inequality to obtain

$$\begin{aligned}
 \Pr(|Y - \bar{f}| \geq \varepsilon) &\leq \frac{\text{Var}[Y]}{\varepsilon^2} \\
 &= \frac{\text{Var}[(\sum_{i=1}^m f(X_i))/m]}{\varepsilon^2} \\
 &= \frac{\sum_{i=1}^m \text{Var}[f(X_i)]}{m^2 \varepsilon^2} \\
 &\leq \frac{m}{m^2 \varepsilon^2} = \frac{1}{m \varepsilon^2},
 \end{aligned}$$

since  $\text{Var}[f(X_i)] \leq \mathbb{E}[(f(X_i))^2] \leq 1$ . We therefore find  $\Pr(|Y - \bar{f}| \geq \varepsilon) \leq \delta$  when  $m = 1/\delta \varepsilon^2$ . (In fact, one can prove that  $\text{Var}[f(X_i)] \leq 1/4$ , giving a slightly better bound; this is left as Exercise 15.4.)

Using pairwise independent samples requires more samples:  $\Theta(1/\delta \varepsilon^2)$  instead of the  $\Theta(\ln(1/\delta)/\varepsilon^2)$  samples when they are independent. But recall from Section 15.1.3 that we can obtain up to  $2^n$  pairwise independent samples with just  $2n$  uniform independent bits. Hence, as long as  $1/\delta \varepsilon^2 < 2^n$ , just  $2n$  random bits suffice; this is much less than the number required when using completely independent samples. Usually  $\varepsilon$  and  $\delta$  are fixed constants independent of  $n$ , and this type of estimation is quite efficient in terms of both the number of random bits used and the computational cost.

### 15.3. Universal Families of Hash Functions

Up to this point, when studying hash functions we modeled them as being completely random in the sense that, for any collection of items  $x_1, x_2, \dots, x_k$ , the hash values  $h(x_1), h(x_2), \dots, h(x_k)$  were considered uniform and independent over the range of the hash function. This was the framework we used to analyze hashing as a balls-and-bins problem in Chapter 5. The assumption of a completely random hash function simplifies the analysis for a theoretical study of hashing. In practice, however, completely random hash functions are too expensive to compute and store, so the model does not fully reflect reality.

Two approaches are commonly used to implement practical hash functions. In many cases, heuristic or ad hoc functions designed to appear random are used. Although these functions may work suitably for some applications, they generally do not have any associated provable guarantees, making their use potentially risky. Another approach is to use hash functions for which there are some provable guarantees. We trade away the strong statements one can make about completely random hash functions for weaker statements with hash functions that are efficient to store and compute.

We consider one of the computationally simplest classes of hash functions that provide useful provable performance guarantees: universal families of hash functions. These functions are widely used in practice.

**Definition 15.2:** Let  $U$  be a universe with  $|U| \geq n$  and let  $V = \{0, 1, \dots, n-1\}$ . A family of hash functions  $\mathcal{H}$  from  $U$  to  $V$  is said to be  $k$ -universal if, for any elements

$x_1, x_2, \dots, x_k$  and for a hash function  $h$  chosen uniformly at random from  $\mathcal{H}$ , we have

$$\Pr(h(x_1) = h(x_2) = \dots = h(x_k)) \leq \frac{1}{n^{k-1}}.$$

A family of hash functions  $\mathcal{H}$  from  $U$  to  $V$  is said to be strongly  $k$ -universal if, for any elements  $x_1, x_2, \dots, x_k$ , any values  $y_1, y_2, \dots, y_k \in \{0, 1, \dots, n-1\}$ , and a hash function  $h$  chosen uniformly at random from  $\mathcal{H}$ , we have

$$\Pr((h(x_1) = y_1) \cap (h(x_2) = y_2) \cap \dots \cap (h(x_k) = y_k)) = \frac{1}{n^k}.$$

We will primarily be interested in 2-universal and strongly 2-universal families of hash functions. When we choose a hash function from a family of 2-universal hash functions, the probability that any two elements  $x_1$  and  $x_2$  have the same hash value is at most  $1/n$ . In this respect, a hash function chosen from a 2-universal family acts like a random hash function. It does not follow, however, that for 2-universal families the probability of any three values  $x_1, x_2$ , and  $x_3$  having the same hash value is at most  $1/n^2$ , as would be the case if the hash values of  $x_1, x_2$ , and  $x_3$  were mutually independent.

When a family is strongly 2-universal and we choose a hash function from that family, the values  $h(x_1)$  and  $h(x_2)$  are pairwise independent, since the probability that they take on any specific pair of values is  $1/n^2$ . Because of this, hash functions chosen from a strongly 2-universal family are also known as *pairwise independent* hash functions. More generally, if a family is strongly  $k$ -universal and we choose a hash function from that family, then the values  $h(x_1), h(x_2), \dots, h(x_k)$  are  $k$ -wise independent. Notice that a strongly  $k$ -universal hash function is also  $k$ -universal.

To gain some insight into the behavior of universal families of hash functions, let us revisit a problem we considered in the balls-and-bins framework of Chapter 5. We saw in Section 5.2 that, when  $n$  items are hashed into  $n$  bins by a completely random hash function, the maximum load is  $\Theta(\log n / \log \log n)$  with high probability. We now consider what bounds can be obtained on the maximum load when  $n$  items are hashed into  $n$  bins using a hash function chosen from a 2-universal family.

First, consider the more general case where we have  $m$  items labeled  $x_1, x_2, \dots, x_m$ . For  $1 \leq i < j \leq m$ , let  $X_{ij} = 1$  if items  $x_i$  and  $x_j$  land in the same bin. Let  $X = \sum_{1 \leq i < j \leq m} X_{ij}$  be the total number of collisions between pairs of items. By the linearity of expectations,

$$\mathbf{E}[X] = \mathbf{E} \left[ \sum_{1 \leq i < j \leq m} X_{ij} \right] = \sum_{1 \leq i < j \leq m} \mathbf{E}[X_{ij}].$$

Since our hash function is chosen from a 2-universal family, it follows that

$$\mathbf{E}[X_{ij}] = \Pr(h(x_i) = h(x_j)) \leq \frac{1}{n}$$

and hence

$$\mathbf{E}[X] \leq \binom{m}{2} \frac{1}{n} < \frac{m^2}{2n}. \quad (15.1)$$

Markov's inequality then yields

$$\Pr\left(X \geq \frac{m^2}{n}\right) \leq \Pr(X \geq 2\mathbb{E}[X]) \leq \frac{1}{2}.$$

If we now suppose that the maximum number of items in a bin is  $Y$ , then the number of collisions  $X$  must be at least  $\binom{Y}{2}$ . Therefore,

$$\Pr\left(\binom{Y}{2} \geq \frac{m^2}{n}\right) \leq \Pr\left(X \geq \frac{m^2}{n}\right) \leq \frac{1}{2},$$

which implies that

$$\Pr(Y - 1 \geq m\sqrt{2/n}) \leq \frac{1}{2}.$$

In particular, in the case where  $m = n$ , the maximum load is at most  $1 + \sqrt{2n}$  with probability at least  $1/2$ .

This result is much weaker than the one for perfectly random hash functions, but it is extremely general in that it holds for any 2-universal family of hash functions. The result will prove useful for designing perfect hash functions, as we describe in Section 15.3.3.

### 15.3.1. Example: A 2-Universal Family of Hash Functions

Let the universe  $U$  be the set  $\{0, 1, 2, \dots, m-1\}$  and let the range of our hash function be  $V = \{0, 1, 2, \dots, n-1\}$ , with  $m \geq n$ . Consider the family of hash functions obtained by choosing a prime  $p \geq m$ , letting

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod n$$

and then taking the family

$$\mathcal{H} = \{h_{a,b} \mid 1 \leq a \leq p-1, 0 \leq b \leq p-1\}.$$

Notice that  $a$  cannot here take on the value 0.

**Lemma 15.6:**  $\mathcal{H}$  is 2-universal.

*Proof:* We count the number of functions in  $\mathcal{H}$  for which two distinct elements  $x_1$  and  $x_2$  from  $U$  collide.

First we note that, for any  $x_1 \neq x_2$ ,

$$ax_1 + b \neq ax_2 + b \bmod p.$$

This follows because  $ax_1 + b = ax_2 + b \bmod p$  implies that  $a(x_1 - x_2) = 0 \bmod p$ , yet here both  $a$  and  $(x_1 - x_2)$  are nonzero modulo  $p$ .

In fact, for every pair of values  $(u, v)$  such that  $u \neq v$  and  $0 \leq u, v \leq p-1$ , there exists exactly one pair of values  $(a, b)$  for which  $ax_1 + b = u \bmod p$  and  $ax_2 + b = v$

mod  $p$ . This pair of equations has two unknowns, and its unique solution is given by:

$$\begin{aligned} a &= \frac{v - u}{x_2 - x_1} \bmod p, \\ b &= u - ax_1 \bmod p. \end{aligned}$$

Since there is exactly one hash function for each pair  $(a, b)$ , it follows that there is exactly one hash function in  $\mathcal{H}$  for which

$$ax_1 + b = u \bmod p \quad \text{and} \quad ax_2 + b = v \bmod p.$$

Therefore, in order to bound the probability that  $h_{a,b}(x_1) = h_{a,b}(x_2)$  when  $h_{a,b}$  is chosen uniformly at random from  $\mathcal{H}$ , it suffices to count the number of pairs  $(u, v)$ ,  $0 \leq u, v \leq p - 1$ , for which  $u \neq v$  but  $u = v \bmod n$ . For each choice of  $u$  there are at most  $\lceil p/n \rceil - 1$  possible appropriate values for  $v$ , giving at most  $p(\lceil p/n \rceil - 1) \leq p(p - 1)/n$  pairs. Each pair corresponds to one of  $p(p - 1)$  hash functions, so

$$\Pr(h_{a,b}(x_1) = h_{a,b}(x_2)) \leq \frac{p(p - 1)/n}{p(p - 1)} = \frac{1}{n},$$

proving that  $\mathcal{H}$  is 2-universal. ■

### 15.3.2. Example: A Strongly 2-Universal Family of Hash Functions

We can apply ideas similar to those used to construct the 2-universal family of hash functions in Lemma 15.6 to construct strongly 2-universal families of hash functions. To start, suppose that both our universe  $U$  and the range  $V$  of the hash function are  $\{0, 1, 2, \dots, p - 1\}$  for some prime  $p$ . Now let

$$h_{a,b}(x) = (ax + b) \bmod p,$$

and consider the family

$$\mathcal{H} = \{h_{a,b} \mid 0 \leq a, b \leq p - 1\}.$$

Notice that here  $a$  can take on the value 0, in contrast with the family of hash functions used in Lemma 15.6.

**Lemma 15.7:**  $\mathcal{H}$  is strongly 2-universal.

*Proof:* This is entirely similar to the proof of Lemma 15.2. For any two elements  $x_1$  and  $x_2$  in  $U$  and any two values  $y_1$  and  $y_2$  in  $V$ , we need to show that

$$\Pr((h_{a,b}(x_1) = y_1) \cap (h_{a,b}(x_2) = y_2)) = \frac{1}{p^2}.$$

The condition that both  $h_{a,b}(x_1) = y_1$  and  $h_{a,b}(x_2) = y_2$  yields two equations modulo  $p$  with two unknowns, the values for  $a$  and  $b$ :  $ax_1 + b = y_1 \bmod p$  and  $ax_2 + b = y_2 \bmod p$ . This system of two equations and two unknowns has just one solution:

$$\begin{aligned} a &= \frac{y_2 - y_1}{x_2 - x_1} \bmod p, \\ b &= y_1 - ax_1 \bmod p. \end{aligned}$$

Hence only one choice of the pair  $(a, b)$  out of the  $p^2$  possibilities results in  $x_1$  and  $x_2$  hashing to  $y_1$  and  $y_2$ , proving that

$$\Pr((h_{a,b}(x_1) = y_1) \cap (h_{a,b}(x_2) = y_2)) = \frac{1}{p^2},$$

as required. ■

Although this gives a strongly 2-universal hash family, the restriction that the universe  $U$  and the range  $V$  be the same makes the result almost useless; usually we want to hash a large universe into a much smaller range. We can extend the construction in a natural way that allows much larger universes. Let  $V = \{0, 1, 2, \dots, p-1\}$ , but now let  $U = \{0, 1, 2, \dots, p^k-1\}$  for some integer  $k$  and prime  $p$ . We can interpret an element  $u$  in the universe  $U$  as a vector  $\vec{u} = (u_0, u_1, \dots, u_{k-1})$ , where  $0 \leq u_i \leq p-1$  for  $0 \leq i \leq k-1$  and where  $\sum_{i=0}^{k-1} u_i p^i = u$ . In fact, this gives a one-to-one mapping between vectors of this form and elements of  $U$ .

For any vector  $\vec{a} = (a_0, a_1, \dots, a_{k-1})$  with  $0 \leq a_i \leq p-1$ ,  $0 \leq i \leq k-1$ , and for any value  $b$  with  $0 \leq b \leq p-1$ , let

$$h_{\vec{a},b}(u) = \left( \sum_{i=0}^{k-1} a_i u_i + b \right) \bmod p,$$

and consider the family

$$\mathcal{H} = \{h_{\vec{a},b} \mid 0 \leq a_i, b \leq p-1 \text{ for all } 0 \leq i \leq k-1\}.$$

**Lemma 15.8:**  $\mathcal{H}$  is strongly 2-universal.

**Proof:** We follow the proof of Lemma 15.7. For any two elements  $u_1$  and  $u_2$  with corresponding vectors  $\vec{u}_i = (u_{i,0}, u_{i,1}, \dots, u_{i,k-1})$  and for any two values  $y_1$  and  $y_2$  in  $V$ , we need to show that

$$\Pr((h_{\vec{a},b}(u_1) = y_1) \cap (h_{\vec{a},b}(u_2) = y_2)) = \frac{1}{p^2}.$$

Since  $u_1$  and  $u_2$  are different, they must differ in at least one coordinate. Without loss of generality let  $u_{1,0} \neq u_{2,0}$ . For any given values of  $a_1, a_2, \dots, a_{k-1}$ , the condition that  $h_{\vec{a},b}(u_1) = y_1$  and  $h_{\vec{a},b}(u_2) = y_2$  is equivalent to:

$$\begin{aligned} a_0 u_{1,0} + b &= \left( y_1 - \sum_{j=1}^{k-1} a_j u_{1,j} \right) \bmod p \\ a_0 u_{2,0} + b &= \left( y_2 - \sum_{j=1}^{k-1} a_j u_{2,j} \right) \bmod p. \end{aligned}$$

For any given values of  $a_1, a_2, \dots, a_{k-1}$ , this gives a system with two equations and two unknowns (namely,  $a_0$  and  $b$ ), which – as in Lemma 15.8 – has exactly one solution. Hence, for every  $a_1, a_2, \dots, a_{k-1}$ , only one choice of the pair  $(a_0, b)$  out of the  $p^2$

possibilities results in  $u_1$  and  $u_2$  hashing to  $y_1$  and  $y_2$ , proving that

$$\Pr((h_{\bar{a},b}(u_1) = y_1) \cap (h_{\bar{a},b}(u_2) = y_2)) = \frac{1}{p^2},$$

as required. ■

Although we have described both the 2-universal and the strongly 2-universal hash families in terms of arithmetic modulo a prime number, we could extend these techniques to work over general finite fields – in particular, fields with  $2^n$  elements represented by sequences of  $n$  bits. The extension requires knowledge of finite fields, so we just sketch the result here. The setup and proof are exactly the same as for Lemma 15.8 except that, instead of working modulo  $p$ , we perform all arithmetic in a fixed finite field with  $2^n$  elements. We assume a fixed one-to-one mapping  $f$  from strings of  $n$  bits, which can also be thought of as numbers in  $\{0, 1, \dots, 2^n - 1\}$ , to field elements. We let

$$h_{\bar{a},b}(u) = f^{-1} \left( \sum_{i=0}^{k-1} f(a_i) \cdot f(u_i) + f(b) \right),$$

where the  $a_i$  and  $b$  are chosen independently and uniformly over  $\{0, 1, \dots, 2^n - 1\}$  and where the addition and multiplication are performed over the field. This gives a strongly 2-universal hash function with a range of size  $2^n$ .

### 15.3.3. Application: Perfect Hashing

Perfect hashing is an efficient data structure for storing a *static dictionary*. In a static dictionary, items are permanently stored in a table. Once the items are stored, the table is used only for search operations: a search for an item gives the location of the item in the table or returns that the item is not in the table.

Suppose that a set  $S$  of  $m$  items is hashed into a table of  $n$  bins, using a hash function from a 2-universal family and chain hashing. In chain hashing (see Section 5.5.1), items hashed to the same bin are kept in a linked list. The number of operations for looking up an item  $x$  is proportional to the number of items in  $x$ 's bin. We have the following simple bound.

**Lemma 15.9:** *Assume that  $m$  elements are hashed into an  $n$ -bin chain hashing table by using a hash function  $h$  chosen uniformly at random from a 2-universal family. For an arbitrary element  $x$ , let  $X$  be the number of items at the bin  $h(x)$ . Then*

$$\mathbb{E}[X] \leq \begin{cases} m/n & \text{if } x \notin S, \\ 1 + (m-1)/n & \text{if } x \in S. \end{cases}$$

**Proof:** Let  $X_i = 1$  if the  $i$ th element of  $S$  (under some arbitrary ordering) is in the same bin as  $x$  and 0 otherwise. Because the hash function is chosen from a 2-universal family, it follows that

$$\Pr(X_i = 1) \leq 1/n.$$



Then the first result follows from

$$\mathbf{E}[X] = \mathbf{E} \left[ \sum_{i=1}^m X_i \right] = \sum_{i=1}^m \mathbf{E}[X_i] \leq \frac{m}{n},$$

where we have used the universality of the hash function to conclude that  $\mathbf{E}[X_i] \leq 1/n$ . Similarly, if  $x$  is an element of  $S$  then (without loss of generality) let it be the first element of  $S$ . Hence  $X_1 = 1$ , and again

$$\Pr(X_i = 1) \leq 1/n$$

when  $i \neq 1$ . Therefore,

$$\mathbf{E}[X] = \mathbf{E} \left[ \sum_{i=1}^m X_i \right] = 1 + \sum_{i=2}^m \mathbf{E}[X_i] \leq 1 + \frac{m-1}{n}. \quad \blacksquare$$

Lemma 15.9 shows that the average performance of hashing when using a hash function from a 2-universal family is good, since the time to look through a bin of any item is bounded by a small number. For instance, if  $m = n$  then, when searching the hash table for  $x$ , the expected number of items other than  $x$  that must be examined is at most 1. However, this does not give us a bound on the worst-case time of a lookup. Some bin may contain  $\sqrt{n}$  elements or more, and a search for one of these elements requires a much longer lookup time.

This motivates the idea of *perfect hashing*. Given a set  $S$ , we would like to construct a hash table that gives excellent worst-case performance. Specifically, by perfect hashing we mean that only a constant number of operations are required to find an item in a hash table (or to determine that it isn't there).

We first show that perfect hashing is easy if we are given sufficient space for the hash table and a suitable 2-universal family of hash functions.

**Lemma 15.10:** *If  $h \in \mathcal{H}$  is chosen uniformly at random from a 2-universal family of hash functions mapping the universe  $U$  to  $[0, n-1]$  then, for any set  $S \subset U$  of size  $m$ , the probability of  $h$  being perfect is at least  $1/2$  when  $n \geq m^2$ .*

**Proof:** Let  $s_1, s_2, \dots, s_m$  be the  $m$  items of  $S$ . Let  $X_{ij}$  be 1 if the  $h(s_i) = h(s_j)$  and 0 otherwise. Let  $X = \sum_{1 \leq i < j \leq m} X_{ij}$ . Then, as we saw in Eqn. (15.1), the expected number of collisions when using a 2-universal hash function is

$$\mathbf{E}[X] = \mathbf{E} \left[ \sum_{1 \leq i < j \leq m} X_{ij} \right] = \sum_{1 \leq i < j \leq m} \mathbf{E}[X_{ij}] \leq \binom{m}{2} \frac{1}{n} < \frac{m^2}{2n}.$$

Markov's inequality then yields

$$\Pr \left( X \geq \frac{m^2}{n} \right) \leq \Pr(X \geq 2\mathbf{E}[X]) \leq \frac{1}{2}.$$

Hence, when  $n \geq m^2$ , we find  $X < 1$  with probability at least  $1/2$ . This implies that a randomly chosen hash function is perfect with probability at least  $1/2$ .  $\blacksquare$

To find a perfect hash function when  $n \geq m^2$ , we may simply try hash functions chosen uniformly at random from the 2-universal family until we find one with no collisions. This gives a Las Vegas algorithm. On average we need to try at most two hash functions.

We would like to have perfect hashing without requiring space for  $\Omega(m^2)$  bins to store the set of  $m$  items. We can use a two-level scheme that accomplishes perfect hashing using only  $O(m)$  bins. First, we hash the set into a hash table with  $m$  bins using a hash function from a 2-universal family. Some of these bins will have collisions. For each such bin, we provide a second hash function from an appropriate 2-universal family and an entirely separate second hash table. If the bin has  $k > 1$  items in it then we use  $k^2$  bins in the secondary hash table. We have already shown in Lemma 15.10 that with  $k^2$  bins we can find a hash function from a 2-universal family that will give no collisions. It remains to show that, by carefully choosing the first hash function, we can guarantee that the total space used by the algorithm is only  $O(m)$ .

**Theorem 15.11:** *The two-level approach gives a perfect hashing scheme for  $m$  items using  $O(m)$  bins.*

**Proof:** As we showed in Lemma 15.10, the number of collisions  $X$  in the first stage satisfies

$$\Pr\left(X \geq \frac{m^2}{n}\right) \leq \Pr(X \geq 2E[X]) \leq \frac{1}{2}.$$

When  $n = m$ , this implies that the probability of having more than  $m$  collisions is at most  $1/2$ . Using the probabilistic method, there exists a choice of hash function from the 2-universal family in the first stage that gives at most  $m$  collisions. In fact, such a hash function can be found efficiently by trying hash functions chosen uniformly at random from the 2-universal family, giving a Las Vegas algorithm. We may therefore assume that we have found a hash function for the first stage that gives at most  $m$  collisions.

Let  $c_i$  be the number of items in the  $i$ th bin. Then there are  $\binom{c_i}{2}$  collisions between items in the  $i$ th bin, so

$$\sum_{i=1}^m \binom{c_i}{2} \leq m.$$

For each bin with  $c_i > 1$  items, we find a second hash function that gives no collisions using space  $c_i^2$ . Again, for each bin, this hash function can be found using a Las Vegas algorithm. The total number of bins used is then bounded above by

$$m + \sum_{i=1}^m c_i^2 \leq m + 2 \sum_{i=1}^m \binom{c_i}{2} + \sum_{i=1}^m c_i \leq m + 2m + m = 4m.$$

Hence, the total number of bins used is only  $O(m)$ . ■

## 15.4. Application: Finding Heavy Hitters in Data Streams

A router forwards packets through a network. At the end of the day, a natural question for a network administrator to ask is whether the number of bytes traveling from a source  $s$  to a destination  $d$  that have passed through the router is larger than a predetermined threshold value. We call such a source–destination pair a *heavy hitter*.

When designing an algorithm for finding heavy hitters, we must keep in mind the restrictions of the router. Routers have very little memory and so cannot keep a count for each possible pair  $s$  and  $d$ , since there are simply too many such pairs. Also, routers must forward packets quickly, so the router must perform only a small number of computational operations for each packet. We present a randomized data structure that is appropriate even with these limitations. The data structure requires a *threshold*  $q$ ; all source–destination pairs that are responsible for at least  $q$  total bytes are considered heavy hitters. Usually  $q$  is some fixed percentage, such as 1%, of the total expected daily traffic. At the end of the day, the data structure gives a list of possible heavy hitters. All true heavy hitters (responsible for at least  $q$  bytes) are listed, but some other pairs may also appear in the list. Two other input constants,  $\epsilon$  and  $\delta$ , are used to control what extraneous pairs might be put in the list of heavy hitters. Suppose that  $Q$  represents the total number of bytes over the course of the day. Our data structure has the guarantee that any source–destination pair that constitutes less than  $q - \epsilon Q$  bytes of traffic is listed with probability at most  $\delta$ . In other words, all heavy hitters are listed; all pairs that are sufficiently far from being a heavy hitter are listed with probability at most  $\delta$ ; pairs that are close to heavy hitters may or may not be listed.

This router example is typical of many situations where one wants to keep a succinct summary of a large data stream. In most *data stream models*, large amounts of data arrive sequentially in small blocks, and each block must be processed before the next block arrives. In the setting of network routers, each block is generally a packet. The amount of data being handled is often so large and the time between arrivals is so small that algorithms and data structures that use only a small amount of memory and computation per block are required.

We can use a variation of a Bloom filter, discussed in Section 5.5.3, to solve this problem. Unlike our solution there, which assumed the availability of completely random hash functions, here we obtain strong, provable bounds using only a family of 2-universal hash functions. This is important, because efficiency in the router setting demands the use of only very simple hash functions that are easy to compute, yet at the same time we want provable performance guarantees.

We refer to our data structure as a *count-min filter*. The count-min filter processes a sequential stream of pairs  $X_1, X_2, \dots$  of the form  $X_t = (i_t, c_t)$ , where  $i_t$  is an item and  $c_t > 0$  is an integer count increment. In our routing setting,  $i_t$  would be the pair of source–destination addresses of a packet and  $c_t$  would be the number of bytes in the packet. Let

$$\text{Count}(i, T) = \sum_{t: i_t = i, 1 \leq t \leq T} c_t.$$

That is,  $\text{Count}(i, T)$  is the total count associated with an item  $i$  up to time  $T$ . In the routing setting,  $\text{Count}(i, T)$  would be the total number of bytes associated with packets with an address pair  $i$  up to time  $T$ . The count-min filter keeps a running approximation of  $\text{Count}(i, T)$  for all items  $i$  and all times  $T$  in such a way that it can track heavy hitters.

A count-min filter consists of  $m$  counters. We assume henceforth that our counters have sufficiently many bits that we do not need to worry about overflow; in many practical situations, 32-bit counters will suffice and are convenient for implementation. A count-min filter uses  $k$  hash functions. We split the counters into  $k$  disjoint groups  $G_1, G_2, \dots, G_k$  of size  $m/k$ . For convenience, we assume in what follows that  $k$  divides  $m$  evenly. We label the counters by  $C_{a,j}$ , where  $1 \leq a \leq k$  and  $0 \leq j \leq m/k - 1$ , so that  $C_{a,j}$  corresponds to the  $j$ th counter in the  $a$ th group. That is, we can think of our counters as being organized in a 2-dimensional array, with  $m/k$  counters per row and  $k$  columns. Our hash functions should map items from the universe into counters, so we have hash functions  $H_a$  for  $1 \leq a \leq k$ , where  $H_a : U \rightarrow [0, m/k - 1]$ . That is, each of the  $k$  hash functions takes an item from the universe and maps it into a number  $[0, m/k - 1]$ . Equivalently, we can think of each hash function as taking an item  $i$  and mapping it to the counter  $C_{a, H_a(i)}$ . The  $H_a$  should be chosen independently and uniformly at random from a 2-universal hash family.

We use our counters to keep track of an approximation of  $\text{Count}(i, T)$ . Initially, all the counters are set to 0. To process a pair  $(i_t, c_t)$ , we compute  $H_a(i_t)$  for each  $a$  with  $1 \leq a \leq k$  and increment  $C_{a, H_a(i_t)}$  by  $c_t$ . Let  $C_{a,j}(T)$  be the value of the counter  $C_{a,j}$  after processing  $X_1$  through  $X_T$ . We claim that, for any item, the smallest counter associated with that item is an upper bound on its count, and with bounded probability the smallest counter associated with that item is off by no more than  $\varepsilon$  times the total count of all the pairs  $(i_t, c_t)$  processed up to that point. Specifically, we have the following theorem.

**Theorem 15.12:** *For any  $i$  in the universe  $U$  and for any sequence  $(i_1, c_1), \dots, (i_T, c_T)$ ,*

$$\min_{j=H_a(i), 1 \leq a \leq k} C_{a,j}(T) \geq \text{Count}(i, T).$$

*Furthermore, with probability  $1 - (k/m\varepsilon)^k$  over the choice of hash functions,*

$$\min_{j=H_a(i), 1 \leq a \leq k} C_{a,j}(T) \leq \text{Count}(i, T) + \varepsilon \sum_{t=1}^T c_t.$$

**Proof:** The first bound,

$$\min_{j=H_a(i), 1 \leq a \leq k} C_{a,j}(T) \geq \text{Count}(i, T),$$

is trivial. Each counter  $C_{a,j}$  with  $j = H_a(i)$  is incremented by  $c_t$  when the pair  $(i, c_t)$  is seen in the stream. It follows that the value of each such counter is at least  $\text{Count}(i, T)$  at any time  $T$ .

For the second bound, consider any specific  $i$  and  $T$ . We first consider the specific counter  $C_{1, H_1(i)}$  and then use symmetry. We know that the value of this counter is at least  $\text{Count}(i, T)$  after the first  $T$  pairs. Let the random variable  $Z_1$  be the amount the counter is incremented owing to items other than  $i$ . Let  $X_i$  be a random variable that is

1 if  $i_t \neq i$  and  $H_1(i_t) = H_1(i)$ ;  $X_t$  is 0 otherwise. Then

$$Z_1 = \sum_{\substack{t: 1 \leq t \leq T, i_t \neq i \\ H_1(i_t) = H_1(i)}} c_t = \sum_{t=1}^T X_t c_t.$$

Because  $H_1$  is chosen from a 2-universal family, for any  $i_t \neq i$  we have

$$\Pr(H_1(i_t) = H_1(i)) \leq \frac{k}{m}$$

and hence

$$\mathbf{E}[X_t] \leq \frac{k}{m}.$$

It follows that

$$\mathbf{E}[Z_1] = \mathbf{E}\left[\sum_{t=1}^T X_t c_t\right] = \sum_{t=1}^T c_t \mathbf{E}[X_t] \leq \frac{k}{m} \sum_{t=1}^T c_t.$$

By Markov's inequality,

$$\Pr\left(Z_1 \geq \varepsilon \sum_{t=1}^T c_t\right) \leq \frac{k/m}{\varepsilon} = \frac{k}{m\varepsilon}. \quad (15.2)$$

Let  $Z_2, Z_3, \dots, Z_k$  be corresponding random variables for each of the other hash functions. By symmetry, all of the  $Z_i$  satisfy the probabilistic bound of Eqn. (15.2). Moreover, the  $Z_i$  are independent, since the hash functions are chosen independently from the family of hash functions. Hence

$$\Pr\left(\min_{j=1}^k Z_j \geq \varepsilon \sum_{t=1}^T c_t\right) = \prod_{j=1}^k \Pr\left(Z_j \geq \varepsilon \sum_{t=1}^T c_t\right) \quad (15.3)$$

$$\leq \left(\frac{k}{m\varepsilon}\right)^k. \quad (15.4)$$

■

It is easy to check using calculus that  $(k/m\varepsilon)^k$  is minimized when  $k = m\varepsilon/e$ , in which case

$$\left(\frac{k}{m\varepsilon}\right)^k = e^{-m\varepsilon/e}.$$

Of course,  $k$  needs to be chosen so that  $k$  and  $m/k$  are integers, but this does not substantially affect the probability bounds.

We can use a count-min filter to track heavy hitters in the routing setting as follows. When a pair  $(i_T, c_T)$  arrives, we update the count-min filter. If the minimum hash value associated with  $i_T$  is at least the threshold  $q$  for heavy hitters, then we put the item into a list of potential heavy hitters. We do not concern ourselves with the details of performing operations on this list, but note that it can be organized to allow updates

and searches in time logarithmic in its size by using standard balanced search-tree data structures; alternatively, it could be organized in a large array or a hash table.

Recall that we use  $Q$  to represent the total traffic at the end of the day.

**Corollary 15.13:** *Suppose that we use a count-min filter with  $k = \lceil \ln \frac{1}{\delta} \rceil$  hash functions,  $m = \lceil \ln \frac{1}{\delta} \rceil \cdot \lceil \frac{e}{\epsilon} \rceil$  counters, and a threshold  $q$ . Then all heavy hitters are put on the list, and any source–destination pair that corresponds to fewer than  $q - \epsilon Q$  bytes is put on the list with probability at most  $\delta$ .*

**Proof:** Since counts increase over time, we can simply consider the situation at the end of the day. By Theorem 15.12, the count-min filter will ensure that all true heavy hitters are put on the list, since the smallest counter value for a true heavy hitter will be at least  $q$ . Further, by Theorem 15.12, the smallest counter value for any source–destination pair that corresponds to fewer than  $q - \epsilon Q$  bytes reaches  $q$  with probability at most

$$\left( \frac{k}{m\epsilon} \right)^k \leq e^{-\ln(1/\delta)} = \delta. \quad \blacksquare$$

The count-min filter is very efficient in terms of using only limited randomness in its hash functions, only  $O(\frac{1}{\epsilon} \ln \frac{1}{\delta})$  counters, and only  $O(\ln \frac{1}{\delta})$  computations to process each item. (Additional computation and space might be required to handle the list of potential heavy hitters, depending on its representation.)

Before ending our discussion of the count-min filter, we describe a simple improvement known as *conservative update* that often works well in practice, although it is difficult to analyze. When a pair  $(i_t, c_t)$  arrives, our original count-min filter adds  $c_t$  to each counter  $C_{a,j}$  that the item  $i$  hashes to, thereby guaranteeing that

$$\min_{j=H_a(i), 1 \leq a \leq k} C_{a,j}(T) \geq \text{Count}(i, T)$$

holds for all  $i$  and  $T$ . In fact, this can often be guaranteed without adding  $c_t$  to each counter. Consider the state after the  $(t - 1)$ th pair has been processed. Suppose that, inductively, up to that point we have, for all  $i$ ,

$$\min_{j=H_a(i), 1 \leq a \leq k} C_{a,j}(t - 1) \geq \text{Count}(i, t - 1).$$

Then, when  $(i_t, c_t)$  arrives, we need to ensure that

$$C_{a,j}(t) \geq \text{Count}(i_t, t)$$

for all counters, where  $j = H_a(i_t)$ ,  $a \leq 1 \leq k$ . But

$$\text{Count}(i_t, t) = \text{Count}(i_t, t - 1) + c_t \leq \min_{j=H_a(i_t), 1 \leq a \leq k} C_{a,j}(t - 1) + c_t.$$

Hence we can look at the minimum counter value  $v$  obtained from the  $k$  counters that  $i_t$  hashes to, add  $c_t$  to that value, and increase to  $v + c_t$  any counter that is smaller than  $v + c_t$ . An example is given in Figure 15.1. An item arrives with a count of 3; at the time of arrival, the smallest counter associated with the item has value 4. It follows that the count for this item is at most 7, so we can update all associated counters to ensure they are all at least 7. In general, if all the counters  $i_t$  hashes to are equal, conservative

( $i, 3$ )

3	8	5	1
4	8	3	2
3	0	5	9

3	8	5	1
7	8	3	2
3	0	7	9

**Figure 15.1:** An item comes in, and 3 is to be added to the count. The initial state is on the left; the shaded counters need to be updated. Using conservative update, the minimum counter value 4 determines that all corresponding counters need to be pushed up to at least  $4 + 3 = 7$ . The resulting state after the update is shown on the right.

update is equivalent to just adding  $c_i$  to each counter. When the  $i_t$  are not all equal, the conservative update improvement adds less to some of the counters, which will tend to reduce the errors that the filter produces.

## 15.5. Exercises

**Exercise 15.1:** A fair coin is flipped  $n$  times. Let  $X_{ij}$ , with  $1 \leq i < j \leq n$ , be 1 if the  $i$ th and  $j$ th flip landed on the same side; let  $X_{ij} = 0$  otherwise. Show that the  $X_{ij}$  are pairwise independent but not independent.

**Exercise 15.2:** (a) Let  $X$  and  $Y$  be numbers that are chosen independently and uniformly at random from  $\{0, 1, \dots, n\}$ . Let  $Z$  be their sum modulo  $n + 1$ . Show that  $X$ ,  $Y$ , and  $Z$  are pairwise independent but not independent.

(b) Extend this example to give a collection of random variables that are  $k$ -wise independent but not  $(k + 1)$ -wise independent.

**Exercise 15.3:** For any family of hash functions from a finite set  $U$  to a finite set  $V$ , show that, when  $h$  is chosen at random from that family of hash functions, there exists a pair of elements  $x$  and  $y$  such that

$$\Pr(h(x) = h(y)) \geq \frac{1}{|V|} - \frac{1}{|U|}.$$

This result should not depend on how the function  $h$  is chosen from the family.

**Exercise 15.4:** Show that, for any discrete random variable  $X$  that takes on values in the range  $[0, 1]$ ,  $\text{Var}[X] \leq 1/4$ .

**Exercise 15.5:** Suppose we have a randomized algorithm *Test* for testing whether a string appears in a language  $L$  that works as follows. Given an input  $x$ , the algorithm *Test* chooses a random integer  $r$  uniformly from the set  $S = \{0, 1, \dots, p - 1\}$  for some prime  $p$ . If  $x$  is in the language, then  $\text{Test}(x, r) = 1$  for at least half of the possible values of  $r$ . A value of  $r$  such that  $\text{Test}(x, r) = 1$  is called a *witness* for  $x$ . If  $x$  is not in the language, then  $\text{Test}(x, r) = 0$  always.

If we run the algorithm *Test* twice on an input  $x \in L$  by choosing two numbers  $r_1$  and  $r_2$  independently and uniformly from  $S$  and evaluating  $\text{Test}(x, r_1)$  and  $\text{Test}(x, r_2)$ , then we find a witness with probability at least  $3/4$ . Argue that we can obtain a witness with probability at least  $1 - 1/t$  using the same amount of randomness by letting  $s_i = r_1 i + r_2 \bmod p$  and evaluating  $\text{Test}(x, s_i)$  for values  $0 \leq i \leq t < p$ .

**Exercise 15.6:** Our analysis of Bucket sort in Section 5.2.2 assumed that  $n$  elements were chosen independently and uniformly at random from the range  $[0, 2^k)$ . Suppose instead that  $n$  elements are chosen uniformly from the range  $[0, 2^k)$  in such a way that they are only pairwise independent. Show that, under these conditions, Bucket sort still requires linear expected time.

**Exercise 15.7:** (a) We have shown that the maximum load when  $n$  items are hashed into  $n$  bins using a hash function chosen from a 2-universal family of hash functions is at most  $1 + \sqrt{2n}$  with probability at least  $1/2$ . Generalize this argument to  $k$ -universal hash functions. That is, find a value such that the probability that the maximum load is larger than that value is at most  $1/2$ .

(b) In Lemma 5.1 we showed that, under the standard balls-and-bins model, the maximum load when  $n$  balls are thrown independently and uniformly at random into  $n$  bins is at most  $3 \ln n / \ln \ln n$  with probability  $1 - 1/n$ . Find the smallest value of  $k$  such that the maximum load is at most  $3 \ln n / \ln \ln n$  with probability at least  $1/2$  when choosing a hash function from a  $k$ -universal family.

**Exercise 15.8:** We can generalize the problem of finding a large cut to finding a large  $k$ -cut. A  $k$ -cut is a partition of the vertices into  $k$  disjoint sets, and the value of a cut is the weight of all edges crossing from one of the  $k$  sets to another. In Section 15.1.2 we considered 2-cuts when all edges had the same weight 1, and we showed how to derandomize the standard randomized algorithm using collections of  $n$  pairwise independent bits. Explain how this derandomization could be generalized to obtain a polynomial time algorithm for 3-cuts, and give the running time for your algorithm. (*Hint:* You may want to use a hash function of the type found in Section 15.3.2.)

**Exercise 15.9:** Suppose we are given  $m$  vectors  $\bar{v}_1, \bar{v}_2, \dots, \bar{v}_m \in \{0, 1\}^\ell$  such that any  $k$  of the  $m$  vectors are linearly independent modulo 2. Let  $\bar{v}_i = (v_{i,1}, v_{i,2}, \dots, v_{i,\ell})$ . Let  $\bar{u}$  be chosen uniformly at random from  $\{0, 1\}^\ell$ , and let  $X_i = \sum_{j=1}^{\ell} v_{i,j} u_j \bmod 2$ . Show that the  $X_i$  are uniform,  $k$ -wise independent bits.

**Exercise 15.10:** We examine a specific way in which 2-universal hash functions differ from completely random hash functions. Let  $S = \{0, 1, 2, \dots, k\}$ , and consider a hash function  $h$  with range  $\{0, 1, 2, \dots, p-1\}$  for some prime  $p$  much larger than  $k$ . Consider the values  $h(0), h(1), \dots, h(k)$ . If  $h$  is a completely random hash function, then the probability that  $h(0)$  is smaller than any of the other values is roughly  $1/(k+1)$ . (There may be a tie for the smallest value, so the probability that any  $h(i)$  is the unique smallest value is slightly less than  $1/(k+1)$ .) Now consider a hash function  $h$  chosen



uniformly from the family

$$\mathcal{H} = \{h_{a,b} \mid 0 \leq a, b \leq p-1\}$$

of Section 15.3.2. Estimate the probability that  $h(0)$  is smaller than  $h(1), \dots, h(k)$  by randomly choosing 10,000 hash functions from  $\mathcal{H}$  and computing  $h(x)$  for all  $x \in S$ . Run this experiment for  $k = 32$  and  $k = 128$ , using primes  $p = 5,023,309$  and  $p = 10,570,849$ . Is your estimate close to  $1/(k+1)$ ?

**Exercise 15.11:** In a multi-set, each element can appear multiple times. Suppose that we have two multi-sets,  $S_1$  and  $S_2$ , consisting of positive integers. We want to test if the two sets are the “same” – that is, if each item appears the same number of times in each set. One way of doing this is to sort both sets and then compare the sets in sorted order. This takes  $O(n \log n)$  time if each multi-set contains  $n$  elements.

- (a) Consider the following algorithm. Hash each element of  $S_1$  into a hash table with  $cn$  counters; the counters are initially 0, and the  $i$ th counter is incremented each time the hash value of an element is  $i$ . Using another table of the same size and using the same hash function, do the same for  $S_2$ . If the  $i$ th counter in the first table matches the  $i$ th counter in the second table for all  $i$ , report that the sets are the same, and otherwise report that the sets are different.

Analyze the running time and error probability of this algorithm, assuming that the hash function is chosen from a 2-universal family. Explain how this algorithm can be extended to a Monte Carlo algorithm, and analyze the trade-off between its running time and its error probability.

- (b) We can also design a Las Vegas algorithm for this problem. Now each entry in the hash table corresponds to a linked list of counters. Each entry holds a list of the number of occurrences of each element that hashes to that location; this list can be kept in sorted order. Again, we create a hash table for  $S_1$  and a hash table for  $S_2$ , and we test after hashing if the resulting tables are equal.

Argue that this algorithm requires only linear expected time using only linear space.

**Exercise 15.12:** In Section 15.3.1 we showed that the family

$$\mathcal{H} = \{h_{a,b} \mid 1 \leq a \leq p-1, 0 \leq b \leq p-1\}$$

is 2-universal when  $p \geq n$ , where

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod n.$$

Consider now the hash functions

$$h_a(x) = (ax \bmod p) \bmod n$$

and the family

$$\mathcal{H}' = \{h_a \mid 1 \leq a \leq p-1\}.$$

Give an example to show that  $\mathcal{H}'$  is not 2-universal. Then prove that  $\mathcal{H}'$  is almost 2-universal in the following sense: for any  $x, y \in \{0, 1, 2, \dots, p-1\}$ , if  $h$  is chosen uniformly at random from  $\mathcal{H}'$  then

$$\Pr(h(x) = h(y)) \leq \frac{2}{n}.$$

**Exercise 15.13:** In describing count-min filters, we assumed that the data stream consisted of pairs of the form  $(i_t, c_t)$ , where  $i_t$  was an item and  $c_t > 0$  an integer count increment. Suppose that one were also allowed to decrement the count for an item, so that the stream could include pairs of the form  $(i_t, c_t)$  with  $c_t < 0$ . We could require that the total count for an item  $i$ ,

$$\text{Count}(i, T) = \sum_{t: i_t=i, 1 \leq t \leq T} c_t,$$

always be positive.

Explain how you could modify or otherwise use count-min filters to find heavy hitters in this situation.