Fast second moment estimation and other fun with Mersenne primes

Mikkel Thorup

August 7, 2020

Abstract

The classic way of computing a k-universal hash function is to use a random degree-$(k-1)$ polynomial over a prime field $\mathbb{Z}_p$. For a fast computation of the polynomial, the prime p is often chosen as a Mersenne prime $p = 2^b - 1$.

In this paper, we show that there are other nice advantages to using Mersenne primes. Our view is that the output of the hash function is a b-bit integer that is uniformly distributed in $[2^b]$, except that p (the all `1`s value) is missing. Thinking of the hash values as almost uniform b-bit integers leads to simple efficient code with strong theoretical qualities. We will demonstrate this with focus on the 4-universal hashing in the classic count-sketch for second moment estimation.

Contents

1 Introduction **2**

1.1 Polynomial hashing using Mersenne primes 2

1.1.1 Good bucketing with powers of two 3

1.2 Two-for-one hash functions in second moment estimation 5

1.2.1 Second moment estimation 5

1.2.2 Two-for-one hash functions with b-bit hash values 6

1.2.3 Two-for-one hashing with Mersenne primes 7

1.3 Arbitrary number of buckets 8

1.4 Even faster modulo with Mersenne Primes 9

2 Analysis of second moment estimation using Mersenne primes **10**

2.1 Analysis in the classic case 10

2.2 The real analysis of two-for-one using Mersenne primes 11

3 Algorithms and analysis with arbitrary number of buckets **14**

3.1 Collision probability with most uniform distributions 14

3.2 Two-for-one hashing from uniform bits to arbitrary number of buckets 15

3.3 Two-for-one hashing from Mersenne primes to arbitrary number of buckets 16

1 Introduction

Polynomial hashing using Mersenne primes is well-known for being faster than polynomial hashing using arbitrary primes. Here we argue that uniform hash values from a Mersenne prime field with prime $p = 2^b - 1$ can largely be treated as uniform b-bit strings, that is, we can use the tool box of very simple and efficient tricks for uniform b-bit strings.

From $[2^b]$ we are missing p, the all `1`s value, but a careful analysis shows that this has only minor effect on the quality of the outcome. Many of the ideas presented here would not apply at all if we had a prime $2^b - a$ with $a > 1$, e.g., $a = 3$, so our work is very particular to Mersenne primes.

To put our results in perspective, suppose we were hashing n keys uniformly into b-bit strings. The probability that any of them hash to $p = 2^b - 1$ is at most n/p. This means that any error probability we might have proved assuming uniform b-bit hash values is off by at most n/p. This may be good if p/n is sufficiently large. However, the analysis we present yields errors that below n/p^2 which is good if p is large even if n is close to p.

1.1 Polynomial hashing using Mersenne primes

The classic definition of k-universal (or k-independent) hashing goes back to Carter and Wegman [**?**].

Definition 1 *A random hash function $h : U \to R$ is k-universal if for any $j \leq k$ distinct keys $x_0, \ldots, x_{j-1}$, the vector $(h(x_0), \ldots, h(x_{j-1}))$ is uniformly distributed in R^j. This is equivalent to saying that each $h(x_i)$ is uniform in R and that all the $h(x_i)$ are indepent of each other.*

We generally use the notation $[s] = \{0, \ldots, s-1\}$. The classic example of k-universal hash function is uniformly random degree-$(k-1)$ polynomial over a prime field $\mathbb{Z}_p$, that is, we pick a uniformly random vector $\vec{a} = (a_0, \ldots, a_{k-1}) \in \mathbb{Z}_p^k$ of k coefficients, and define $h_{\vec{a}} : [p] \to [p]$, by

$$h_{\vec{a}}(x) = \left(\sum_{x \in [k]} a_i x^i \right) \bmod p.$$

Often we are given a key domain $[u]$ and a range of hash values $[r]$. Assuming $p \geq \max\{u, r\}$, we define $h_{\vec{a}}^r : [u] \to [r]$, by

$$h_{\vec{a}}^r(x) = h_{\vec{a}}(x) \bmod r.$$

With this definition, the hash values of k distinct keys remain independent, but they are not exactly uniform. More precisely, for any $i \in [r]$, we have that $\Pr[h_{\vec{a}}^r(x) = i]$ is either $\lfloor p/r \rfloor / p$ or $\lceil p/r \rceil / p$. In either case, this is $(1 \pm r/p)/r$, so this is close to uniform if $p \gg r$.

The main problem with the above approach is speed because computing the mod-function is slow on most computers. Often it is OK if we for the hash range $[r]$ pick $r = 2^\ell$ as a power of two. Then $r - 1$ consists of ℓ `1`s, and then

$$y \bmod r = y \mathbin{\&} (r-1)$$

where & denotes bit-wise AND. Bit-wise operations are extremely fast, so now $\bmod r$ is not an issue.

For general primes, computing $\bmod p$ is slow, but an old idea, mentioned by Carter and Wegmen [**?**], is to use a Mersenne prime $p = 2^b - 1$, e.g., $p = 2^{61} - 1$ for hashing 32-bit keys or $p = 2^{89} - 1$ for hashing 64-bit keys. The point in using such a Mersenne prime is that

$$y \equiv y \bmod 2^b + \lfloor y/2^b \rfloor \pmod p. \tag{1}$$

This leads to efficient computations because

$$y \bmod 2^b = y\texttt{\&}p \quad \text{and} \quad \lfloor y/2^b \rfloor = y\texttt{>>}b.$$

Here the bit-wise AND (&) and the right-shift (>>) are both very fast operations. For an extra fast implementation, we further assume that $p = 2^b - 1 \geq 2u - 1$. This is automatically satisfied in the typical case where u is a power of two, e.g., 2^{32} or 2^{64}. This is all used in Algorithm 1 to make a very fast computation of a degree $(k-1)$-polynomial using Horner's rule. Note that the reduction $y \leftarrow (y\texttt{\&}p) + (y\texttt{>>}b)$ is applied after every multiplication so that the numbers involved never get too big (multiplication of large numbers is expensive). The above completes our description of

Algorithm 1: For $x \in [u]$, prime $p = 2^b - 1 \geq 2u - 1$, and $\vec{a} = (a_0, \ldots, a_{k-1}) \in [p]^k$, compute $h_{\vec{a}}(x) = \left(\sum_{i \in [q]} a_i x^i\right) \bmod p$

```
$y \leftarrow a_{k-1}$;
for $i = q-2, \ldots, 0$ do                          ▷  $y < 2p$
    $y \leftarrow yx + a_i$                   ▷  $y < 2p(u-1) + (p-1) < (2u-1)p \leq p^2$
    $y \leftarrow (y\&p) + (y>>b)$                    ▷  $y < p + p^2/2^b < 2p$
if $y \geq p$ then $y \leftarrow y - p$;
return $y$
```

how Mersenne primes are normally used for fast computation of k-universal hash functions. We shall in fact show how it can be made even faster, but the main point in this paper is to present an analysis, showing how our k-universal hash values from $[2^b - 1]$ can be almost as good as if they were uniformly distributed b-bit strings (we are only missing the all `1`s value $2^b - 1$).

We note that $k = 2$, we do have the fast multiply-shift scheme of Dietzfelbinger [**?**], that directly gives 2-universal hashing from b-bit strings to ℓ-bit strings, but for $k > 2$, there is no faster method that can be implemented in with portable code in a standard programming language like C.

1.1.1 Good bucketing with powers of two

As a first trivial illustration of the quality that we get using a Mersenne prime $p = 2^b - 1$, consider the case mentioned above where we want hash values in the range $[r]$ where $r = 2^\ell$ is a power of two. We will often refer to the hash values in $[r]$ as buckets so that we are hashing keys to buckets.

Avoiding a degenerate case, we assume $r > 1$. In particular this implies that r does not divide our prime p.

We assume a k-universal hash function $h : [u] \to [p]$, e.g., the one from Algorithm 1. To get a hash values in $[r]$, we defined $h^r : [u] \to [r]$ by

$$h^r(x) = h(x) \bmod r = h(x) \,\&\, (r-1).$$

As discussed previously, the hash values of up to k distinct keys remain independent with h^r. The issue is that hash values from h^r are not quite uniform in $[r]$.

Recall that for any key x, we have $h(x)$ uniformly distributed in $[2^b - 1]$. This is the uniform distribution on b-bit strings except that we are missing $p = 2^b - 1$. Now p is the all `1`s, and $p \,\&\, (r-1) = r - 1$. Therefore, for any any $i < r - 1$,

$$\Pr[h^r(x) = i] = \lceil p/r \rceil / p = ((p+1)/r)/p = (1 + 1/p)/r, \tag{2}$$

while

$$\Pr[h^r(x) = r-1] = \lfloor p/r \rfloor / p = ((p+1-r)/r)/p = (1 - (r-1)/p)/r. \tag{3}$$

Thus $\Pr[h^r(x) = i] \le (1 + 1/p)/r$ for all $i \in [r]$. This upper-bound only has a relative error of $1/p$ from the uniform $1/r$. For comparison, if we had used a prime of the form $p = 2^b - a$ and $a < r$, then we would only get an upper bound of $(1 + a/p)/r$. Below we return to a Mersenne prime $p = 2^b - 1$

Combining (2) and (3) with pairwise independence, for any distinct keys $x, y \in [u]$, we get collision probability

$$\begin{aligned}\Pr[h^r(x) = h^r(y)] &= \sum_{i \in [r]} \Pr[h^r(x) = h^r(y) = i] = \sum_{i \in [r]} \Pr[h^r(x) = i]^2 \\ &= (r-1)((1+1/p)/r)^2 + ((1-(r-1)r/p)/r)^2 \\ &= \frac{r + (r^2 - r)/p^2}{r^2} = (1 + (r-1)/p^2)/r < (1 + r/p^2)/r. \end{aligned} \tag{4}$$

We note that relative error r/p^2 is small as long as p is large.

Selecting arbitrary bits from the hash value Interestingly, the above analysis holds no matter which ℓ bits we use when mapping the hash values from $[2^b - 1]$ to $[2^\ell]$. Let $\mu : [2^b] \to [2^\ell]$ be any map defined by selecting ℓ bits from a b-bit string. Above we used $\mu(y) = y \bmod 2^\ell = y \,\&\, (2^\ell - 1)$, selecting the ℓ least significant bits, but we could also use $\mu(y) = \lfloor y/2^{b-\ell} \rfloor = y \texttt{>>} (b - \ell)$ selecting the ℓ most significant bits. The basic point is that a uniform distribution on $[2^b]$ maps to a uniform distribution on $[2^\ell]$. We are only missing the all `1`s value $p = 2^b - 1$ which maps to $2^\ell - 1$ regardless of which ℓ bits we select, so our equations (2)–(4) hold no matter which ℓ bits we select for h^r.

The fact that it doesn't matter which ℓ bits we select is only true because we use a Mersenne prime $p = 2^b - 1$. Suppose we used some other b-bit prime $p = 2^b - a$ where $2^{b-\ell} < a < 2^{b-1}$. If we select the ℓ most signifant bits, then 0 elements from $[p]$ map to $2^\ell - 1$ while $2^{b-\ell}$ elements from $[p]$ map to 0. However, with the ℓ least significant bits, we have $\lfloor p/2^\ell \rfloor$ or $\lceil p/2^\ell \rceil$ elements from $[p]$ mapping to each element in $[2^\ell]$, so the maximal difference is 1.

1.2 Two-for-one hash functions in second moment estimation

In this section we discuss how we can get several hash functions for the price of one, and apply the idea to second moment estimation using count sketches [**?**].

Suppose we had a k-universal hash function into b-bit strings. We note that using standard programming languages such as C, we have no simple and efficient method computing such hash functions when $k > 2$. However, later we will argue that polynomial hashing using a Mersenne prime $2^b - 1$ delivers a better-than-expected approximation.

Let $h : U \to [2^b]$ be k-universal. By definition this means that if we have $j \leq k$ distinct keys $x_1, \ldots, x_j$, then $(h(x_1), \ldots, h(x_j))$ is uniform in $[2^b]^j \equiv [2]^{bj}$, so this means that *all* the bits in $h(x_1), \ldots, h(x_j)$ are independent and uniform. We can use this to split our b-bit hash values into smaller segments, and sometimes use them as if they were the output of independently computed hash functions. We illustrate this idea below in the context of the second moment estimation.

1.2.1 Second moment estimation

We now review the second moment estimation of streams based on count sketches [**?**] (which are based on the celebrated second moment AMS-estimator from [**?**])

The basic set-up is as follows. For keys in $[u]$ and integer values in $\mathbb{Z}$, we are given a stream of key/value $(x_0, \Delta_0), \ldots, (x_{n-1}, \Delta_{n-1}) \in [u] \times \mathbb{Z}$. The total value of key $x \in [u]$ is

$$f_x = \sum_{i \in [n], x_i = x} \Delta_i.$$

We let $n \leq u$ be the number of non-zero values $f_x \neq 0$, $x \in [u]$. Often n is much smaller than u. We define the mth moment $F_m = \sum_{x \in [u]} f_y^m = \|f\|_m^m$. The goal here is to estimate the second moment $F_2 = \sum_{x \in [u]} f_x^2$ which is the square of the Euclidean norm $\|f\|_2$.

Algorithm 2: Count Sketch. Uses a vector/array C of r integers and two independent 4-universal hash functions $i : [u] \to [r]$ and $s : [u] \to \{-1, 1\}$. .

```
Initialize  For $i \in [t]$, set $C[i] \leftarrow 0$.
Process$(x, \Delta)$  $C[i(x)] \leftarrow C[i(x)] + s(x)\Delta$.
Output  $X = \sum_{i \in [t]} C[i]^2$.
```

The standard analysis [**?**] shows that

$$\mathsf{E}[X] = F_2 \tag{5}$$

$$\mathsf{Var}[X] = 2(F_2 - F_4)/r < 2F_2/r \tag{6}$$

By Chebyshev's inequality, this implies

$$\Pr[|X - F_2| \geq \varepsilon F_2] \leq \mathsf{Var}[X]/(\varepsilon F_2)^2 < 2/(k\varepsilon^2).$$

With $t = 8/\varepsilon^2$, the error probability is below 1/4. To reduce the error probability, we can use the standard trick of making r independent experiments and return the median estimate. Using Chernoff bounds, the probability that the median deviates by more than εF_2 is bounded by $\exp(-r/12)$.

1.2.2 Two-for-one hash functions with b-bit hash values

As the count sketch is described above, it uses two independent 4-universal hash functions $i : [u] \to [r]$ and $s : [u] \to \{-1, 1\}$, but 4-universal hash functions are generally slow to compute, so, aiming to save roughly a factor 2 in speed, a tempting idea is to compute them both using a single hash function.

The analysis behind (5) and (6) does not quite require $i : [u] \to [r]$ and $s : [u] \to \{-1, 1\}$ to be indepedent. It suffices that the hash values are uniform and that for any given set of $j \leq 4$ distinct keys $x_1, \ldots, x_j$, the $2j$ hash values $i(x_1), \ldots, i(x_j), s(x_1), \ldots, s(x_j)$ are independent. A critical step in the analysis is that if A depends on $i(x_1), \ldots, i(x_j), s(x_2), \ldots, s(x_j)$, but not on $s(x_1)$, then

$$\mathsf{E}[s(x_1)A] = 0.$$

This follows because $\mathsf{E}[s(x_1)] = 0$ by uniformity of $s(x_1)$ and because $s(x_1)$ is independent of A.

Assuming that $t = 2^\ell$ is a power of two, we can easily construct $i : [u] \to [t]$ and $s : [u] \to \{-1, 1\}$ using a single 4-universal hash function $h : [u] \to [2^b]$ where $b > \ell$. Recall that all the bits in $h(x_1), \ldots, h(x_4)$ are independent. We can therefore use the ℓ least significant bits of $h(x)$ for $i(x)$ and the most significant bit of $h(x)$ for a bit $a(x) \in [2]$, and finally set $s(x) = 1 - 2a(x)$. The construction is summarized in Algorithm 3

Algorithm 3: For key $x \in [u]$, compute $i(x) = i_x \in [2^\ell]$ and $s(x) = s_x \in \{-1, +1\}$, using $h : [u] \to [2^b]$ where $b > \ell$.

```
$h_x \leftarrow h(x)$                  ▷ $h_x$ uses $b$ bits
$i_x \leftarrow h_x \& (2^\ell - 1)$   ▷ $i_x$ gets $\ell$ least significant bits of $h_x$
$a_x \leftarrow h_x >> (b-1)$          ▷ $a_x$ gets the most significant bit of $h_x$
$s_x \leftarrow 1 - 2a_x$              ▷ $a_x \in [2]$ is converted to a sign $s_x \in \{-1, +1\}$
```

Lemma 1.1 *Suppose $h : [u] \to [2^b]$ is k-universal. Let $i : [u] \to [2^\ell]$ and $s : [u] \to \{-1, 1\}$ be constructed from h as described in Algorithm 3. Then h and s are both k-universal. Moreover, for any $j \leq k$ distinct keys $x_1, \ldots, x_j$, the $2j$ hash values $i(x_1), \ldots, i(x_j), s(x_1), \ldots, s(x_j)$ are independent. In particular, if A depends on $i(x_1), \ldots, i(x_j), s(x_2), \ldots, s(x_j)$, but not on $s(x_1)$, then*

$$\mathsf{E}[s(x_1)A] = 0 \tag{7}$$

Note that Algorithm 3 is well defined as long as h returns a b-bit integer. However, Lemma 1.1 requires that h is k-universal into $[2^b]$, which in particular implies that the hash values are uniform in $[2^b]$.

1.2.3 Two-for-one hashing with Mersenne primes

Above we discussed how useful it would be with k-universal hashing mapping uniformly into b-bit strings. The issue was that the lack of efficient implementations with standard portable code if $k > 2$. However, when $2^b - 1$ is a Mersenne prime $p \geq u$, then we do have have the efficient computation from Algorithm 1 of a k-universal hash function $h : [u] \to [2^b - 1]$. The hash values are b-bit integers, and they are uniformly distributed, except that we are missing the all `1`s value $p = 2^b - 1$. We want to understand how this missing value affects us if we try to split the hash values as in Algorithm 3. Thus, we assume a k-universal hash function $h : [u] \to [2^b - 1]$ from which we construct $i : [u] \to [2^\ell]$ and $s : [u] \to \{-1, 1\}$ as described in Algorithm 3. As usual, we assume $2^\ell > 1$. Since i_x and s_x are both obtained by selection of bits from h_x, we know from Section 1.1.1 that each of them have close to uniform distributions. However, we need a good replacement for (7) which besides uniformity, requires i_x and s_x to be independent, and this is certainly not the case.

Before getting into the analysis, we argue that we really do get two hash functions for the price of one. The point is that our efficient computation in Algorithm 1 requires that we use a Mersenne prime $2^b - 1$ such that $u \leq 2^{b-1}$, and this is even if our final target is to produce just a single bit for the sign function $s : [u] \to \{-1, 1\}$. We also know that $2^\ell < u$, for otherwise we get perfect results implementing $i : [u] \to [2^\ell]$ as the identifty function (perfect because it is collision free). Thus we can assume $\ell < b$, hence that h provides enough bits for both s and i.

We now consider the effect of the hash values from h being uniform in $[2^b - 1]$ instead of in $[2^b]$. Suppose we want to compute the expected value of an expression B depending only on the independent hash values $h(x_1), \ldots, h(x_j)$ of $j \leq k$ distinct keys $x_1, \ldots, x_j$.

Our generic idea is to play with the distribution of $h(x_1)$ while leaving the distributions of the other independent hash values $h(x_2) \ldots, h(x_j)$ unchanged, that is, they remain uniform in $[2^b - 1]$. We will consider having $h(x_1)$ uniformly distributed in $[2^b]$, denoted $h(x_1) \leftarrow U[2^b]$, but then we later have to subtract the "fake" case where $h(x_1) = p = 2^b - 1$. Making the distribution of $h(x_1)$ explicit, we get

$$\begin{aligned}\mathop{\mathsf{E}}_{h(x_1) \leftarrow U[p]}[B] &= \sum_{y \in [p]} \mathop{\mathsf{E}}_{h(x_1)=y}[B]/p = \sum_{y \in [2^b]} \mathop{\mathsf{E}}_{h(x_1)=y}[B]/p - \mathop{\mathsf{E}}_{h(x_1)=p}[B]/p \\ &= \mathop{\mathsf{E}}_{h(x_1) \leftarrow U[2^b]}[B](p+1)/p - \mathop{\mathsf{E}}_{h(x_1)=p}[B]/p. \qquad (8)\end{aligned}$$

Let us now apply this idea our situation where $i : [u] \to [2^\ell]$ and $s : [u] \to \{-1, 1\}$ are constructed from h as described in Algorithm 3. We will prove

Lemma 1.2 *Consider distinct keys $x_1, \ldots, x_j$, $j \leq k$ and an expression $B = s(x_1)A$ where A depends on $i(x_1), \ldots, i(x_j)$ and $s(x_2), \ldots, s(x_j)$ but not $s(x_1)$. Then*

$$\mathsf{E}[s(x_1)A] = \mathsf{E}[A \mid i(x_1) = 2^\ell - 1]/p. \qquad (9)$$

Proof When $h(x_1) \leftarrow U[2^b]$, then $s(x_1)$ is uniform in $\{-1, +1\}$ and independent of $i(x_1)$. The remaining $(i(x_i), s(x_i))$, $i > 1$, are independent of $s(x_1)$ because they are functions of $h(x_i)$ which is independent of $h(x_1)$, so we conclude that

$$\mathsf{E}_{h(x)\leftarrow U[p+1]}[s(x_1)A] = 0$$

Finally, when $h(x_1) = p$, we get $s(x_1) = -1$ and $i(x_1) = 2^\ell - 1$, so applying (8), we conclude that

$$\mathsf{E}[s(x_1)A] = \mathsf{E}_{i(x)=2^\ell-1}[A]/p.$$

■

Above (9) is our replacement for (7), that is, when the hash values from h are uniform in $[2^b - 1]$ instead of in $[2^b]$, then $\mathsf{E}[s(x_1)B]$ is reduced by $\mathsf{E}_{i(x)=2^\ell-1}[B]/p$. For large p, this is a small additive error. Using this in a careful analysis, we will show that our fast second moment estimation based on Mersenne primes performes almost perfectly:

Theorem 1.3 *Let $r > 1$ and $u > r$ be powers of two and let $p = 2^b - 1 > u$ be a Mersenne prime. Suppose with have a k-universal hash function $h : [u] \to [2^b - 1]$, e.g., generated using Algorithm 1. Suppose $i : [u] \to [r]$ and $s : [u] \to \{-1, 1\}$ are constructed from h as described in Algorithm 3. Using this i and s in the Count Sketch Algorithm 2, the second moment estimate $X = \sum_{i\in[k]} C_i^2$ satsifies:*

$$\mathsf{E}[X] = F_2 + (F_1^2 - F_2)/p^2 < (1 + n/p^2)\, F_2, \tag{10}$$
$$\mathsf{Var}[X] < 2(F_2^2 - F_4)/r + F_2^2(2.33 + 4n/r)/p^2 < 2F_2^2/r. \tag{11}$$

The difference from (5) and (6) is negligiable when p is large. Theorem 1.3 will be proved in Section 2.

1.3 Arbitrary number of buckets

We now consider the general case where we want to hash into a set of buckets R whose size is not a power of two. Suppose we have a 2-universal hash function $h : U \to Q$. We will compose h with a map $\mu : Q \to R$, and use $\mu \circ h$ as a hash function from U to R.

Let $q = |Q|$ and $r = |R|$. We want the map μ to be *most uniform* in the sense that for bucket $i \in R$, the number of elements from Q mapping to i is either $\lfloor q/r \rfloor$ or $\lceil q/r \rceil$. Then the uniformity of hash values with h implies for any key x and bucket $i \in R$

$$\lfloor q/r \rfloor / q \le \Pr[\mu \circ h(x) = i] \le \lceil q/r \rceil / q.$$

Below we typically have $Q = [q]$ and $R = [r]$. A standard example of a most uniform map $\mu : [q] \to [r]$ is $\mu(x) = x \bmod r$ which the one used above when we defined $h^r : [u] \to [r]$, but as we mentioned before, the modulo operation is quite slow unless r is a power of two.

Another example of a most uniform map $\mu : [q] \to [r]$ is $\mu(x) = \lfloor xr/q \rfloor$, which is also quite slow in general, but if $q = 2^b$ is a power of two, it can be implemented as $\mu(x) = (xr) >> b$ where $>>$ denotes right-shift. This would be yet another advantage of of having k-universal hashing into $[2^b]$.

Now, our interest is the case where q is a Mersenne prime $p = 2^b - 1$. We want an efficient and most uniform map $\mu : [2^b - 1]$ into any given $[r]$. Our simple solution is to define

$$\mu(x) = \lfloor (x+1)r/2^b \rfloor = ((x+1)r){>>}b. \tag{12}$$

Lemma 1.4 (iii) below states that (12) indeed gives a most uniform map.

Lemma 1.4 *Let r and b the positive integers $2^b - 1$.*

(i) $x \mapsto (xr){>>}b$ is a most uniform map from $[2^b]$ to $[r]$.

(ii) $x \mapsto (xr){>>}b$ is a most uniform map from $[2^b] \setminus \{0\} = \{1, \ldots, 2^b - 1\}$ to $[r]$.

(iii) $x \mapsto ((x+1)r){>>}b$ is a most uniform map from $[2^b - 1]$ to $[r]$.

Proof Trivially (ii) implies (iii). The statement (i) is folklore and easy to prove, so we know that every $i \in [r]$ gets hit by $\lfloor 2^b/r \rfloor$ or $\lceil 2^b/r \rceil$ elements from $[2^b]$. It is also clear that $\lceil 2^b/r \rceil$ elements, including 0, maps to 0. To prove (ii), we remove 0 from $[2^b]$, implying that only $\lceil 2^b/r \rceil - 1$ elements map to 0. For all positive integers q and r, $\lceil (q+1)/r \rceil - 1 = \lfloor q/r \rfloor$, and we use this here with $q = 2^b - 1$. It follows that all buckets from $[r]$ gets $\lfloor q/r \rfloor$ or $\lfloor q/r \rfloor + 1$ elements from $Q = \{1, \ldots, q\}$. If r does not divide q then $\lfloor q/r \rfloor + 1 = \lceil q/r \rceil$, as desired. However, if r divides q, then $\lfloor q/r \rfloor = q/r$, and this is the least number of elements from Q hitting any bucket in $[r]$. Then no bucket from $[r]$ can get hit by more than $q/r = \lceil q/r \rceil$ elements from Q. This completes the proof of (ii), and hence of (iii). ■

We note that our trick does not work when $q = 2^b - a$ for $a \geq 2$, that is, using $x \mapsto ((x+a)r){>>}b$, for in this general case, the number of elements hashing to 0 is $\lceil 2^b/r \rceil - a$, or 0 if $a \geq \lfloor 2^b/r \rfloor$. Our new uniform map from (12) is thus very specific to Mersenne prime fields.

We shall see in Section **??** that our new uniform map works very well in conjunction with the the idea of splitting of hash values values from Section 1.2.3.

1.4 Even faster modulo with Mersenne Primes

We even suggest a speedup the computation of $\mathrm{mod} p$ for Mersenne primes $p = 2^b - 1$. The issue in Algorithm 1 is that if-statement can be slow because of branch prediction. From the standard

Algorithm 4: For Mersenne prime $p = 2^b - 1$ and $x \leq p^2$, computes $y = x \bmod p$ and $x = \lfloor x/p \rfloor$

Algorithm 1 this means that we can replaces the last two statements $y \leftarrow (y \& p) + (y{>>}b)$; if $y \geq p$ then $y \leftarrow y - p$

2 Analysis of second moment estimation using Mersenne primes

In this section, we will prove Theorem 1.3. Thus we have $r = 2^\ell > 1$ and $u > r$ both powers of two and $p = 2^b - 1 > u$ a Mersenne prime.

Now let us set up the relevant variables for the basic count sketch. For each key $x \in [u]$, we have a value $f_x \in \mathbb{Z}$, and the goal was to estimate the second moment $F_2 = \sum_{x \in u} f^2$.

We had two functions $i : [u] \to [r]$ and $s : [u] \to \{-1, +1\}$. For notational convinience, we define $i_x = i(x)$ and $s_x = s(x)$.

For each $i \in [r]$, we have a counter $C_i = \sum_{x \in [u]} s_x f_x [i_x = i]$, and we define the estimator $X = \sum_{i \in [k]} C_i^2$. We want to study how well it approximates F_2. We have

$$\begin{aligned} X &= \sum_{i \in [r]} \left(\sum_{x \in [u]} s_x f_x [i_x = i] \right)^2 \\ &= \sum_{i \in [r]} \sum_{x, y \in [u]} s_x f_x [i_x = i = i_y] s_y f_y \\ &= \sum_{x, y \in [u]} s_x f_x [i_x = i_y] s_y f_y. \\ &= \sum_{x \in [u]} f_x^2 + \sum_{x, y \in [u], x \neq y} s_x f_x [i_x = i_y] s_y f_y. \end{aligned}$$

Thus

$$X = F_2 + Y \text{ where } Y = \sum_{x, y \in [u], x \neq y} s_x f_x [i_x = i_y] s_y f_y. \tag{13}$$

We thus want to provide bounds on the error Y.

2.1 Analysis in the classic case

First, as a warm-up for later comparison, we analyze the simple classic case where $i : [u] \to [2^\ell]$ and $s : [u] \to \{-1, 1\}$ are independent 4-universal hash functions. In this case, we first argue that

$$\mathsf{E}[Y] = 0. \tag{14}$$

To prove (14), we argue that each term of Y from (13) has 0 expected value. With $x \neq y$, by 2-universality (implied by 4-universality) and independence of s and h, we have that s_x is independent of s_y, i_x, and i_y. Moreover, $\mathsf{E}[s_x] = 0$, so

$$\mathsf{E}[s_x f_x [i_x = i_y] s_y f_y] = \mathsf{E}[s_x] \, \mathsf{E}[f_x [i_x = i_y] s_y f_y] = 0.$$

It follows that $\mathsf{E}[Y] = 0$, completing the proof of (14).

Next we want to bound the variance of X which is the same as that of Y since F_2 is a constant. Since $\mathsf{E}[Y] = 0$, we have

$$\begin{aligned}\mathsf{Var}[X] = \mathsf{Var}[Y] = \mathsf{E}[Y^2] = \mathsf{E}[(&\sum_{x,y\in[u],x\neq y} s_x f_x [i_x = i_y] s_y f_y)^2] \\ = \sum_{x,y,x',y'\in[u],x\neq y,x'\neq y'} &\mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})].\end{aligned}$$

Consider one of the terms $\mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})]$. We have $x \neq y, x' \neq y'$. Suppose that any one of the keys, say x, is unique, that is, $x \notin \{y, x', y'\}$. Then, by 4-universality, s_x is independent of s_y, $s_{x'}$, and $s_{y'}$. Morever, it is independent of the hash function h. Since $\mathsf{E}[s_x] = 0$, we get

$$\begin{aligned}&\mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})] \\ =&\, \mathsf{E}[s_x]\, \mathsf{E}[(f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})] = 0.\end{aligned}$$

Thus we can restict our attention to terms with no unique keys. Since $x \neq y$ and $x' \neq y'$, we conclude that $(x, y) = (x', y')$ or $(x, y) = (y', x')$. Therefore

$$\begin{aligned}\mathsf{Var}[X] &= \sum_{x,y,x',y'\in[u],x\neq y,x'\neq y'} \mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})] \\ &= 2 \sum_{x,y\in[u],x\neq y,(x',y')=(x,y)} \mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})] \\ &= 2 \sum_{x,y\in[u],x\neq y} \mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)^2] \\ &= 2 \sum_{x,y\in[u],x\neq y} \mathsf{E}[(f_x^2 f_y^2 [i_x = i_y])] \\ &= 2 \sum_{x,y\in[u],x\neq y} (f_x^2 f_y^2)/r \\ &= 2(F_2^2 - F_4)/r.\end{aligned}$$

This completes the proof of (6). In the above analysis, we did not need s and i to be completely independent. All we needed was that for any $j \leq 4$ distinct keys $x_1, \ldots, x_j$, the hash values $s(x_1), \ldots, s(x_j)$ and $i(x_1), \ldots, i(x_j)$ are all independent and uniform in the desired domain. This was why we could use a single k-universal hash function $h : [u] \to [2^b]$ with $b > \ell$, and use it to construct $s : [u] \to \{-1, +1\}$ and $i : [u] \to [2^\ell]$ as described in Algorithm 3 (c.f. Lemma 1.1).

2.2 The real analysis of two-for-one using Mersenne primes

We now consider the case where the functions $s : [u] \to \{-1, +1\}$ and $i : [u] \to [2^\ell]$ are constructed as described in Algorithm 3 from a single k-universal hash function $h : [u] \to [2^b - 1]$. Here h could be implemented efficiently as in Algorithm 1. As noted earlier, the function i is the same as h^r with $r = 2^\ell$ in Section 1.1.1, so it satisfies (2)–(4). Also, we have the power of Lemma 1.2 to handle the correlation between s and i.

As above, for notational convinience, we let s_x denote $s(x)$ and i_x denote $i(x)$. Also, recall that $n \leq u$ is the number of non-zero values $f_x \neq 0$, $x \in [u]$.

Expectancy Recall from (13) that we have

$$X = F_2 + Y \text{ where } Y = \sum_{x,y \in [u], x \neq y} s_x f_x [i_x = i_y] s_y f_y.$$

This equality holds regardless of random choices, hence regardless of how h and s are implemented.

To bound $\mathsf{E}[Y]$ we study the expectancy of an arbitrary term, that is, for some $x, y \in [u], x \neq y$, we study $\mathsf{E}[s_x f_x [i_x = i_y] s_y f_y]$. Using Lemma 1.2, we get

$$\begin{aligned} \mathsf{E}[s_x f_x [i_x = i_y] s_y f_y] &= \mathsf{E}[f_x [i_x = i_y] s_y f_y \mid i_x = r-1]/p. \\ &= \mathsf{E}[s_y f_x [r-1 = i_y] f_y]/p. \\ &= \mathsf{E}[f_x [r-1 = i_y] f_y \mid i_y = r-1]/p^2. \\ &= \mathsf{E}[f_x f_y]/p^2 = f_x f_y / p^2. \end{aligned} \tag{15}$$

Thus

$$\mathsf{E}[Y] = \sum_{x,y \in [u], x \neq y} f_x f_y / p^2 = (F_1^2 - F_2)/p^2.$$

Trivially is the equality from (10). For the upper bound, we note that

$$F_1^2 \leq n F_2 \tag{16}$$

This follows because the "variance" over the $|f_x|$ is $F_2 - F_1^2/n = \sum_{x \in [u], f_x \neq 0} (f_x - F_1/n)^2 \geq 0$. Therefore

$$\mathsf{E}[Y] = (F_1^2 - F_2)/p^2 \leq F_2 n / p^2.$$

This completes the proof of (10).

Variance Same method is applied to the analysis of the variance, which is

$$\mathsf{Var}[X] = \mathsf{Var}[Y] \leq \mathsf{E}[Y^2] = \sum_{x,y,x',y' \in [u], x \neq y, x' \neq y'} \mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})].$$

Consider any term in the sum. Suppose some key, say x, is unique in the sense that $x \notin \{y, x', y'\}$. Then we can apply Lemma 1.2. Given that $x \neq y$ and $x' \neq y'$, we have either 2 or 4 such unique keys. If all 4 keys are distinct, as in (15), we get

$$\begin{aligned} &\mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y)(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})] \\ &\qquad = \mathsf{E}[(s_x f_x [i_x = i_y] s_y f_y])] \, \mathsf{E}[(s_{x'} f_{x'} [i_{x'} = i_{y'}] s_{y'} f_{y'})] \\ &\qquad = (f_x f_y / p^2)(f_{x'} f_{y'} / p^2) = f_x f_y f_{x'} f_{y'} / p^4. \end{aligned}$$

The expected sum over all such terms is thus bounded as

$$\sum_{\text{distinct } x,y,x',y'\in[u]} \mathsf{E}[(s_x f_x[i_x=i_y]s_y f_y)(s_{x'}f_{x'}[i_{x'}=i_{y'}]s_{y'}f_{y'})]$$
$$= \sum_{\text{distinct } x,y,x',y'\in[u]} f_x f_y f_{x'} f_{y'}/p^4 < F_1^4/p^4 \le F_2^2 n^2/p^4. \tag{17}$$

The last inequalities used (16). We also have to consider all the cases with two unique keys, e.g., x and x' unique while $y=y'$. Then using Lemma 1.2 and (3), we get

$$\begin{aligned}
&\mathsf{E}[(s_x f_x[i_x=i_y]s_y f_y)(s_{x'}f_{x'}[i_{x'}=i_{y'}]s_{y'}f_{y'})]\\
&\qquad= f_x f_{x'} f_y^2\,\mathsf{E}[s_x s_{x'}[i_x=i_{x'}=i_y]]\\
&\qquad= f_x f_{x'} f_y^2\,\mathsf{E}[s_{x'}[r-1=i_{x'}=i_y]]/p\\
&\qquad= f_x f_{x'} f_y^2\,\mathsf{E}[r-1=i_y]/p^2\\
&\qquad< f_x f_{x'} f_y^2/(rp^2).
\end{aligned}$$

Summing over all terms with x and x' unique while $y=y'$, and using (16) and $u\le p$, we get

$$\sum_{\text{distinct } x,x',y} f_x f_{x'} f_y^2/(rp^2) < F_1^2 F_2/(rp^2) \le F_2^2 n/(rp^2).$$

There are four ways we can pick the two unique keys $a\in\{x,y\}$ and $b\in\{x',y'\}$, so we conclude that

$$\sum_{\substack{x,y,x',y'\in[u],\,x\ne y,\,x'\ne y',\\ \text{two keys are unique}}} \mathsf{E}[(s_x f_x[i_x=i_y]s_y f_y)(s_{x'}f_{x'}[i_{x'}=i_{y'}]s_{y'}f_{y'})] \le 4F_2^2 n/(rp^2) \tag{18}$$

Finally, we need to reconsider the terms with two pairs, that is where $(x,y)=(x',y')$ or $(x,y)=(y',x')$. In this case, $(s_x f_x[i_x=i_y]s_y f_y)(s_{x'}f_{x'}[i_{x'}=i_{y'}]s_{y'}f_{y'}) = f_x^2 f_y^2[i_x=i_y]$. By (2) and (3), we get

$$\begin{aligned}
&\sum_{\substack{x,y,x',y'\in[u],\,x\ne y,\,x'\ne y',\\ (x,y)=(x',y')\,\vee\,(x,y)=(y',x')}} \mathsf{E}[(s_x f_x[i_x=i_y]s_y f_y)(s_{x'}f_{x'}[i_{x'}=i_{y'}]s_{y'}f_{y'})]\\
&\qquad= 2\sum_{x,y\in[u],x\ne y}(f_x^2 f_y^2)\Pr[i_x=i_y]\\
&\qquad= 2\sum_{x,y\in[u],x\ne y}(f_x^2 f_y^2)(1+r/p^2)/r\\
&\qquad= 2(F_2^2-F_4)(1+r/p^2)/r
\end{aligned} \tag{19}$$

Adding up add (17), (18), and (19), we get

$$\mathsf{Var}[Y] \leq 2(F_2^2 - F_4)/r + F_2^2(n^2/p^4 + 2/p^2 + 4n/(rp^2)).$$

We have defined our parameters to satisfy $2 \leq r \leq u/2 \leq (p+1)/4$ where r, u, and $p+1$ are all powers of two. Also $n \leq u$, so it follows that

$$n/p \leq u/n \geq\geq 7/4 \text{ and } p/r \geq 7/2. \tag{20}$$

In particular, $(n/p)^2 \leq (4/7)^2 < 0.33$, so we conclude that

$$\mathsf{Var}[Y] < 2(F_2^2 - F_4)/r + F_2^2(2.33 + 4n/r)/p^2. \tag{21}$$

Next we note that $F_4 \geq F_2^2/n$ follows from (16) applied to vector of squared values $(f_0^2, \ldots, f_{u-1}^2)$. Combined with (21) and (20), we get

$$\begin{aligned}\mathsf{Var}[Y] &< 2(F_2^2 - F_4)/r + F_2^2(2.33 + 4n/r)/p^2 \\ &\leq 2F_2^2/r + F_2^2(-2 + 2.33nr/p^2 + 4n^2/p^2) \\ &< 2F_2^2/r.\end{aligned}$$

This completes the proof of (11) and hence of Theorem 1.3.

3 Algorithms and analysis with arbitrary number of buckets

We now consider the case where we want to hash into a number of buckets. We will analyze the collision probability with most uniform maps introduced in Section 1.3, and later we will show how it can be used in connection with the two-for-one hashing from Section 1.2.3.

3.1 Collision probability with most uniform distributions

We have a hash function $h : U \to Q$, but we want hash values in R, so we need a map $\mu : Q \to R$, and then use $\mu \circ h$ as our hash function from U to R. We normally assume that the hash values with h are pairwise independent, that is, for any distinct x and y, the hash values $h(x)$ and $h(y)$ are independent, but then $\mu(h(x))$ and $\mu(h(y))$ are also independent. This means that the collision probability can be calculated as

$$\Pr[\mu(h(x)) = \mu(h(y))] = \sum_{i \in R} \Pr[\mu(h(x)) = \mu(h(y)) = i] = \sum_{i \in R} \Pr[\mu(h(x) = i)]^2.$$

This sum of squared probabilities attains is minimum value $1/|R|$ exactly when $\mu(h(x))$ is uniform in R.

Let $q = |Q|$ and $r = |R|$. Suppose that h is 2-universal. Then $h(x)$ is uniform in Q, and then we get the lowest collision probability with $\mu \circ h$ if μ is most uniform as defined in Section 1.3, that is, the number of elements from Q mapping to any $i \in [r]$ is either $\lfloor q/r \rfloor$ or $\lceil q/r \rceil$. To calculate the collision probability, Let $a \in [r]$ be such that r divides $q+a$. Then the map μ maps $\lceil q/r \rceil = (q+a)/r$ balls to $r-a$ buckets and $\lfloor q/r \rfloor = (q+a-r)/r$ balls to a buckets. For a key $x \in [u]$, we thus have $r-a$ buckets hit with probability $(1+a/q)/r$ and a buckets hit with probability $(1-(r-a)/q)/r$. The collision probability is then

$$\begin{aligned}\Pr[\mu(h(x)) =&\mu(h(y))] = (r-a)((1+a/q)/r)^2 + a((1-(r-a)r/q)/r)^2\\ &= \frac{(r-a)+(r-a)2a/q+(r-a)a^2/q^2+a-a2(r-a)/p+a(r-a)^2/q^2}{r^2}\\ &= \frac{r+ra(r-a)/q^2}{r^2} = (1+a(r-a)/q^2)/r \le \left(1+(r/(2q))^2\right)/r. \qquad (22)\end{aligned}$$

Note that the above calculation generalizes the one for (4) which had $a=1$. We will think of $(r/(2q))^2$ as the general relative rounding cost when we do not have any information about how r divides q.

3.2 Two-for-one hashing from uniform bits to arbitrary number of buckets

We will now briefly discuss how would get the two-for-one hash functions in count sketches with an arbitrary number r of bins based on a single 4-universal hash function $h : [u] \to [2^b]$. We want to construct the two hash functions $s : [u] \to \{-1,+1\}$ and $i : [u] \to [r]$. As usual the results with uniform b-bit strings will set the bar that we later compare with when from h we get hash values that are only uniform in $[2^b - 1]$.

The construction of s and i is presented in Algorithm 5. The difference relative to Algorithm 3

Algorithm 5: For key $x \in [u]$, compute $i(x) = i_x \in [r]$ and $s(x) = s_x \in \{-1,+1\}$.
Uses 4-universal $h : [u] \to [2^b]$.

```
$h_x \leftarrow h(x)$                              ▷ $h_x$ has $b$ uniform bits
$j_x \leftarrow h_x \& (2^{b-1}-1)$               ▷ $j_x$ gets $b-1$ least significant bits of $h_x$
$i_x \leftarrow (rj_x)$>>$(b-1)$                   ▷ $i_x$ is most uniform in $[r]$
$a_x \leftarrow h_x$>>$(b-1)$                      ▷ $a_x$ gets the most significant bit of $h_x$
$s_x \leftarrow 2b_x - 1$                          ▷ $s_x$ is uniform in $\{-1,+1\}$ and independent of $i_x$.
```

is the computation of i_x where we now first pick out the $(b-1)$-bit string j_x from h_x, and then apply the most uniform map (rj_x)>>$(b-1)$ to get i_x. This does not affect s_x which remains independent of i_x.

We now have to study the effect on our estimate error

$$Y = X - F_2 \sum_{x,y \in [u], x \neq y} s_x f_x [i_x = i_y] s_y f_y.$$

The expectation is exactly as before. The only change is in the analysis of the variance where we before had each i_x uniform in $[r] = [2^\ell]$, hence $\Pr[i_x = i_y] = 1/r$. Now have values uniform in $[2^{b-1}]$ mapped most uniformly to $[r]$ for an arbitrary r. Then by (22), we get $\Pr[i_x = i_y] = \left(1+(r/(2q))^2\right)/r$ where $q = 2^{b-1}$. This increases the variance bound accordingly from $2F_2^2/r$ to

$$\mathsf{Var}[X] = \mathsf{Var}[Y] \le 2F_2^2\left(1+(r/2^b)^2\right)/r. \qquad (23)$$

3.3 Two-for-one hashing from Mersenne primes to arbitrary number of buckets

We will now show how wan get the two-for-one hash functions in count sketches with an arbitrary number r of bins based on a single 4-universal hash function $h : [u] \to [2^b - 1]$. Again we want to construct the two hash functions $s : [u] \to \{-1, +1\}$ and $i : [u] \to [r]$. The construction will be the same as we had in Algorithm 5 when h returned uniform values in $[2^b]$ with the change that we set $h_x \leftarrow h(x) + 1$, so that it becomes uniform in $[2^b] \setminus \{0\}$. It is also convinient to swap the sign of the signbit s_x setting $s_x \leftarrow 2a_x + 1$ instead of $s_x \leftarrow 1 - 2a_x$. The basic reason is that we have swapped the role of `0` and `1` in a_x. The resulting algorithm is presented as Algorithm 6. The rest

Algorithm 6: For key $x \in [u]$, compute $i(x) = i_x \in [r]$ and $s(x) = s_x \in \{-1, +1\}$.
Uses 4-universal $h : [u] \to [p]$ for Mersenne prime $p = 2^b - 1 \geq u$.

```
$h_x \leftarrow h(x) + 1$                    ▷ $h_x$ uses $b$ bits uniformly except $h_x \neq 0$
$j_x \leftarrow h_x \& (2^{b-1} - 1)$          ▷ $j_x$ gets $b - 1$ least significant bits of $h_x$
$i_x \leftarrow (r j_x) >\!> b - 1$            ▷ $i_x$ is quite uniform in $[r]$
$a_x \leftarrow h_x >\!> (b - 1)$              ▷ $a_x$ gets the most significant bit of $h_x$
$s_x \leftarrow 1 - 2b_x$      ▷ $s_x$ is quite uniform in $\{-1, +1\}$ and quite independent of $i_x$.
```

of Algorithm 6 is exactly like Algorithm 5, and we will now discuss the new distributions of the resulting variables. We had h_x uniform in $[2^b] \setminus \{0\}$, and then we set $j_x \leftarrow h_x \& (2^{b-1} - 1)$. Then $j_x \in [2^{b-1}]$ with $\Pr[j_x = 0] = 1/(2^b - 1)$ while $\Pr[j_x = j] = 2/(2^b - 1)$ for all $j > 0$.

Next we set $i_x \leftarrow (r j_x) >\!> b - 1$. We know from Lemma 1.4 (i) that this is a most uniform map from $[2^{b-1}]$ to $[r]$. It maps a maximal number of elements from $[2^{b-1}]$ to 0, including 0 which had half probability for j_x. We conclude

$$\Pr[i_x = 0] = (\lceil 2^{b-1}/r \rceil 2 - 1)/(2^b - 1) \tag{24}$$

$$\Pr[i_x \neq 0] \in \{\lfloor 2^{b-1}/r \rfloor 2/(2^b - 1), \lceil 2^{b-1}/r \rceil 2/(2^b - 1)\}. \tag{25}$$

We note that the probability for 0 is in the middle of the two other bounds and often this yields a more uniform distribution on $[r]$ than the most uniform distribution we could get from the uniform distribution on $[2^{b-1}]$.

With more careful calculations, we can get some nicer bounds that we shall later.

Lemma 3.1 *For any distinct $x, y \in [u]$,*

$$\Pr[i_x = 0] \leq (1 + r/2^b)/r \tag{26}$$

$$\Pr[i_x = i_y] \leq \left(1 + (r/2^b)^2\right)/r. \tag{27}$$

Proof The proof of (26) is a simple calculation. Using (24) and the fact $\lceil 2^{b-1}/r \rceil \le (2^{b-1} + r - 1)/r$ we have

$$\begin{aligned}\Pr[i_x = 0] &\le (2(2^{b-1} + r - 1)/r) - 1)/(2^b - 1) \\ &= ((2^b + r - 2)/r)/(2^b - 1) \\ &= \left(1 + (r-1)/(2^b - 1)\right)/r \\ &\le \left(1 + r/2^b\right)/r.\end{aligned}$$

The last inequality follows because $r < u < 2^b$.

For 27, let $q = 2^{b-1}$ and $p = 1/(2q-1)$. We define $a \ge 0$ to be the smallest integer, such that $r \setminus q + a$. In particular this means $\lceil q/r \rceil = (q+a)/r$ and $\lfloor q/r \rfloor = (q - r + a)/r$.

We bound the sum

$$\Pr[i_x = i_y] = \sum_{k=0}^{r-1} \Pr[i_x = k]^2$$

by splitting into three cases: 1) The case $i_x = 0$, where $\Pr[i_x = 0] = (2\lceil q/r \rceil - 1)p$, 2) the $r - a - 1$ indices j where $\Pr[i_x = j] = 2\lceil q/r \rceil p$, and 3) the a indices j st. $\Pr[i_x = j] = 2\lfloor q/r \rfloor p$.

$$\begin{aligned}\Pr[i_x = i_y] &= (2p\lceil q/r \rceil - p)^2 + (r - a - 1)(2p\lceil q/r \rceil)^2 + (r-a)(2p\lfloor q/r \rfloor)^2 \\ &= ((4a+1)r + 4(q+a)(q-a-1))p^2/r \\ &\le (1 + (r^2 - r)/(2q-1)^2)/r.\end{aligned}$$

The last inequality comes from maximizing over a, which yields $a = (r-1)/2$.

The result now follows from

$$(r^2 - r)/(2q-1)^2 \le (r - 1/2)^2/(2q-1)^2 \le (r/(2q))^2 \tag{28}$$

which holds exactly when $r \le q$.

■

Lemma 3.1 above is all we need to know about the marginal distribution of i_x. However, we also need a replacement for Lemma 1.2 for handling the signbit s_x.

Lemma 3.2 *Consider distinct keys $x_1, \dots, x_j$, $j \le k$ and an expression $B = s_{x_1} A$ where A depends on $i_{x_1}, \dots, i_{x_j}$ and $s_{x_2}, \dots, s_{x_j}$ but not s_{x_1}. Then*

$$\mathsf{E}[s_x A] = \mathsf{E}[A \mid i_x = 0]/p. \tag{29}$$

Proof The proof follows the same idea as that for Lemma 1.2. First we have

$$\mathsf{E}[B] = \mathop{\mathsf{E}}_{h_{x_1} \leftarrow U([2^b]\setminus\{0\})}[B] = \mathop{\mathsf{E}}_{h_{x_1} \leftarrow U[2^b]}[B] 2^b/p - \mathop{\mathsf{E}}_{h_{x_1} = 0}[B]/p.$$

With $h_{x_1} \leftarrow U[2^b]$, the bit a_{x_1} is uniform and independent of j_{x_1}, so $s_{x_1} \in \{-1, +1\}$ is uniform and independent of i_{x_1}, and therefore

$$\mathop{\mathsf{E}}_{h_{x_1} \leftarrow U[2^b]}[s_{x_1} A] = 0.$$

Moreover, $h_{x_1} = 0$ implies $j_x = x_1$, $i_{x_1} = 0$, $a_{x_1} = 0$, and $s_{x_1} = -1$, so

$$\mathsf{E}[s_{x_1}A] = -\mathop{\mathsf{E}}_{h_{x_1}=0}[s_{x_1}A]/p = \mathop{\mathsf{E}}_{i_{x_1}=0}[A].$$

■

We now consider our F_2 extimator

$$X = \sum_{i\in[r]}\left(\sum_{x\in[u]} s_x f_x[i_x = i]\right)^2 = F_2 + Y \text{ where } Y = \sum_{x,y\in[u],x\neq y} s_x f_x[i_x = i_y]s_y f_y.$$

Using Lemma 3.2 instead of Lemma 3.2 we get an almost identical calculation of the expection $\mathsf{E}[Y]$ as in Section 1.2.3 and with the same conclusion that

$$\mathsf{E}[Y] = (F_1^2 - F_2)/p^2 \le (n-1)F_2/p^2. \tag{30}$$

Therfore we still have the same bound (10) as in Theorem 1.3.

Concerning the variance, we have some more changes to the calculations in Section 1.2.3. We still start with

$$\mathsf{Var}[X] = \mathsf{Var}[Y] \le \mathsf{E}[Y^2] = \sum_{x,y,x',y'\in[u],x\neq y,x'\neq y'} \mathsf{E}[(s_x f_x[i_x = i_y]s_y f_y)(s_{x'} f_{x'}[i_{x'} = i_{y'}]s_{y'} f_{y'})].$$

For terms with 4 distinct keys, we still get the same expectation as in Section 1.2.3, so their total contribution to the variance is still the $F_2^2 n^2/p^4$ from (17)

However, for the terms with two unique keys and one pair of identical keys, the factor $\mathsf{E}[i_x = r-1] < 1/r$ gets replaced with $\mathsf{E}[i_x = 0]$ which by (26) in Lemma 3.1 is bound by $(1 + r/2^b)/r$. As a result, for the total contribution of these terms, we have to multiply the $4F_2^2 n/(rp^2)$ from (18) by $(1 + r/2^b)$, so they now sum up to

$$4(1 + r/2^b)F_2^2 n/(rp^2) \tag{31}$$

Finally, for the terms with two pairs of identical keys, $\Pr[i_x = i_y]$ was bounded by $(1 + 1/p)/r$, which is replaced by the bound $\left(1 + (r/2^b)^2\right)/r$, so so

$$\begin{aligned}
2 \sum_{x,y\in[u],x\neq y} &(f_x^2 f_y^2)\Pr[i_x = i_y] \\
&= 2 \sum_{x,y\in[u],x\neq y} (f_x^2 f_y^2)(1 + (r/2^b)^2)/k \\
&= 2(F_2^2 - F_4)(1 + (r/2^b)^2)/k \\
&\le 2(1 - 1/n)F_2^2(1 + (r/2^b)^2)/k
\end{aligned} \tag{32}$$

Adding it all up, we have proved that

$$\mathsf{Var}[Y] \le F_2^2 n^2/p^4 + 2(1 - 1/n)F_2^2(1 + (r/2^b)^2)/r + 4(1 + r/2^b)F_2^2 n/(rp^2).$$

We want the argue that we get the same variance bound as we had in (23) with uniform b-bit hash values; namely that

$$\mathsf{Var}[Y] \leq 2\left(1 + (r/2^b)^2\right) F_2^2/r. \tag{33}$$

which follows if

$$ru^3/p^4 + 4(1 + r/2^b)n^2/p^2 \leq 2.$$

Recall that $2 \leq k < u \leq (p+1)/2$. Since u is a power of two, $u \geq 4$. We conclude $n/p \leq u/p \leq 4/7$, $r/p \leq 3/7$, and $p + 1 \geq 8$. Therefore the left-hand side is bounded by $(3/7)(4/7)^3 + 4(1 + 2/8)(4/7)^2 < 1.8 < 2$. This completes the proof of (33). The basic conclusion is that both with r is a power of two and when r is arbitrary, we get the same variance bounds with Mersenne primes as we did with uniform b-bit strings. The only difference is the small error in expectation from (30). Relative to the correct value F_2, the relative error in the expectation is by a factor of at most n/p^2 which is insignificant when p is large.