Fast hashing with strong concentration bounds

Aamand, Anders; Knudsen, Jakob Bæk Tejs; Knudsen, Mathias Bæk Tejs; Rasmussen, Peter Michael Reichstein; Thorup, Mikkel

Published in:
STOC 2020 - Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing

DOI:
10.1145/3357713.3384259

Publication date:
2020

Document version
Publisher's PDF, also known as Version of record

Fast Hashing with Strong Concentration Bounds

Anders Aamand
aa@di.ku.dk
BARC, University of Copenhagen
Denmark

Jakob Bæk Tejs Knudsen
jakn@di.ku.dk
BARC, University of Copenhagen
Denmark

Mathias Bæk Tejs Knudsen
mathias@tejs.dk
SupWiz
Denmark

Peter Michael Reichstein Rasmussen
pmrr@di.ku.dk
BARC, University of Copenhagen
Denmark

Mikkel Thorup
mikkel2thorup@gmail.com
BARC, University of Copenhagen
Denmark

ABSTRACT

Previous work on tabulation hashing by Pătraşcu and Thorup from STOC'11 on simple tabulation and from SODA'13 on twisted tabulation offered Chernoff-style concentration bounds on hash based sums, e.g., the number of balls/keys hashing to a given bin, but under some quite severe restrictions on the expected values of these sums. The basic idea in tabulation hashing is to view a key as consisting of $c = O(1)$ characters, e.g., a 64-bit key as $c = 8$ characters of 8-bits. The character domain Σ should be small enough that character tables of size $|\Sigma|$ fit in fast cache. The schemes then use $O(1)$ tables of this size, so the space of tabulation hashing is $O(|\Sigma|)$. However, the concentration bounds by Pătraşcu and Thorup only apply if the expected sums are $\ll |\Sigma|$.

To see the problem, consider the very simple case where we use tabulation hashing to throw n balls into m bins and want to analyse the number of balls in a given bin. With their concentration bounds, we are fine if $n = m$, for then the expected value is 1. However, if $m = 2$, as when tossing n unbiased coins, the expected value $n/2$ is $\gg |\Sigma|$ for large data sets, e.g., data sets that do not fit in fast cache.

To handle expectations that go beyond the limits of our small space, we need a much more advanced analysis of simple tabulation, plus a new tabulation technique that we call *tabulation-permutation* hashing which is at most twice as slow as simple tabulation. No other hashing scheme of comparable speed offers similar Chernoff-style concentration bounds.

CCS CONCEPTS

• **Theory of computation** → **Pseudorandomness and derandomization**; *Data structures design and analysis*; *Sketching and sampling*.

STOC '20, June 22–26, 2020, Chicago, IL, USA

ACM ISBN 978-1-4503-6979-4/20/06…$15.00
https://doi.org/10.1145/3357713.3384259

KEYWORDS

hashing, Chernoff bounds, concentration bounds, streaming algorithms, sampling

ACM Reference Format:
Anders Aamand, Jakob Bæk Tejs Knudsen, Mathias Bæk Tejs Knudsen, Peter Michael Reichstein Rasmussen, and Mikkel Thorup. 2020. Fast Hashing with Strong Concentration Bounds. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC '20), June 22–26, 2020, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3357713.3384259

1 INTRODUCTION

Chernoff's concentration bounds [13] date back to the 1950s but bounds of this types go even further back to Bernstein in the 1920s [8]. Originating from the area of statistics they are now one of the most basic tools of randomized algorithms [36]. A canonical form considers the sum $X = \sum_{i=1}^{n} X_i$ of independent random variables $X_1, \ldots, X_n \in [0, 1]$. Writing $\mu = \mathbb{E}[X]$ it holds for any $\varepsilon \geq 0$ that

$$\Pr[X \geq (1+\varepsilon)\mu] \leq \exp(-\mu\, C(\varepsilon)) \leq \exp(-\varepsilon^2\mu/3) \text{ for } \varepsilon \leq 1, \tag{1}$$

and for any $0 \leq \varepsilon \leq 1$

$$\Pr[X \leq (1-\varepsilon)\mu] \leq \exp(-\mu\, C(-\varepsilon)) \leq \exp(-\varepsilon^2\mu/2). \tag{2}$$

Here $C : (-1, \infty) \to [0, \infty)$ is given by $C(x) = (x+1)\ln(x+1) - x$, so $\exp(-C(x)) = \frac{e^x}{(1+x)^{(1+x)}}$. Textbook proofs of (1) and (2) can be found in [36, §4][1]. Writing $\sigma^2 = \mathrm{Var}[X]$, a more general bound is that for any $t \geq 0$,

$$\Pr[|X-\mu| \geq t] \leq 2\exp(-\sigma^2 C(t/\sigma^2)) \leq 2\exp(-(t/\sigma)^2/3) \text{ for } t \leq \sigma^2. \tag{3}$$

Since $\sigma^2 \leq \mu$ and $C(-\varepsilon) \leq 1.5\, C(\varepsilon)$ for $\varepsilon \leq 1$, (3) is at least as good as (1) and (2), up to constant factors, and often better. In this work, we state our results in relation to (3), known as Bennett's inequality [7].

Hashing is another fundamental tool of randomized algorithms dating back to the 1950s [24]. A random hash function, $h : U \to R$,

[1] The bounds in [36, §4] are stated as working only for $X_i \in \{0, 1\}$, but the proofs can easily handle any $X_i \in [0, 1]$.

assigns a hash value, $h(x) \in R$, to every key $x \in U$. Here both U and R are typically bounded integer ranges. The original application was hash tables with chaining where x is placed in bin $h(x)$, but today, hash functions are ubiquitous in randomized algorithms. For instance, they play a fundamental role in streaming and distributed settings where a system uses a hash function to coordinate the random choices for a given key. In most applications, we require concentration bounds for one of the following cases of increasing generality.

(1) Let $S \subseteq U$ be a set of balls and assign to each ball, $x \in S$, a weight, $w_x \in [0,1]$. We wish to distribute the balls of S into a set of bins $R = [m] = \{0, 1, \ldots, m-1\}$. For a bin, $y \in [m]$, $X = \sum_{x \in S} w_x \cdot [h(x) = y]$ is then the total weight of the balls landing in bin y.
(2) We may instead be interested in the total weight of the balls with hash values in the interval $[y_1, y_2)$ for some $y_1, y_2 \in [m]$, that is, $X = \sum_{x \in S} w_x \cdot [y_1 \le h(x) < y_2]$.
(3) More generally, we may consider a fixed *value function* $v \colon U \times R \to [0,1]$. For each key $x \in U$, we define the random variable $X_x = v(x, h(x))$, where the randomness of X_x stems from that of $h(x)$. We write $X = \sum_{x \in U} v(x, h(x))$ for the sum of these values.

To exemplify applications, the first case is common when trying to allocate resources; the second case arises in streaming algorithms; and the third case handles the computation of a complicated statistic, X, on incoming data. In each case, we wish the variable X to be concentrated around its mean, $\mu = \mathbb{E}[X]$, according to the Chernoff-style bound of (3). If we had fully random hashing, this would indeed be the case. However, storing a fully random hash function is infeasible. The goal of this paper is to obtain such concentration with a practical constant-time hash function. More specifically, we shall construct hash functions that satisfy the following definition when X is a random variable as in one of the three cases above.

Definition 1 (Strong Concentration)**.** Let $h \colon [u] \to [m]$ be a hash function, $S \subseteq [u]$ be a set of hash keys of size $n = |S|$, and $X = X(h, S)$ be a random variable, which is completely determined by h and S. Denote by $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$ the expectation and variance of X. We say that X is *strongly concentrated with added error probability* $f(u, n, m)$ if for every $t > 0$,

$$\Pr[|X - \mu| \ge t] \le O\left(\exp\left(-\Omega(\sigma^2 C(t/\sigma^2))\right)\right) + f(u, n, m). \quad (4)$$

Throughout the paper we shall prove properties of random variables that are determined by some hash function. In many cases, we would like these properties to continue to hold while conditioning the hash function on its value on some hash key.

Definition 2 (Query Invariant)**.** Let $h \colon [u] \to [m]$ be a hash function, let $X = X(h)$ be a random variable determined by the outcome of h, and suppose that some property T is true of X. We say that the property is *query invariant* if whenever we choose $x \in [u]$ and $y \in [m]$ and consider the hash function $h' = (h | h(x) = y)$, i.e., h conditioned on $h(x) = y$, property T is true of $X' = X(h')$.

Remark. For example, consider the case (1) from above. We are interested in the random variable $X = \sum_{x \in S} w_x \cdot [h(x) = y]$. Suppose that for every choice of weights, $(w_x)_{x \in S}$, X is strongly concentrated and that this concentration is query invariant. Let $x_0 \in [u]$ be a distinguished query key. Then since for every $y_0 \in [m]$, the hash function $h' = (h | h(x_0) = y_0)$ satisfies that $X' = \sum_{x \in S} w_x \cdot [h'(x) = y_0]$ is strongly concentrated, it follows that $X'' = \sum_{x \in S} w_x \cdot [h(x) = h(x_0)]$ is strongly concentrated. Thus, h allows us to get Chernoff-style concentration on the weight of the balls landing in the same bin as x_0.

This may be generalized such that in the third case from above, the weight function may be chosen as a function of $h(x_0)$. Thus, the property of being query invariant is very powerful. It is worth noting that the constants of the asymptotics may change when conditioning on a query. Furthermore, the expected value and variance of X' may differ from that of X, but this is included in the definition.

One way to achieve Chernoff-style bounds in all of the above cases is through the classic k-independent hashing framework of Wegman and Carter [47]. The random hash function $h : U \to R$ is k-independent if for any k distinct keys $x_1, \ldots, x_k \in U$, $(h(x_1), \ldots, h(x_k))$ is uniformly distributed in R^k. Schmidt and Siegel [42] have shown that with k-independence, the above Chernoff bounds hold with an added error probability decreasing exponentially in k. Unfortunately, a lower bound by Siegel [43] implies that evaluating a k-independent hash function takes $\Omega(k)$ time unless we use a lot of space (to be detailed later).

Pătraşcu and Thorup have shown that Chernoff-style bounds can be achieved in constant time with tabulation based hashing methods; namely simple tabulation [38] for the first case described above and twisted tabulation [41] for all cases. However, their results suffer from some severe restrictions on the expected value, μ, of the sum. More precisely, the speed of these methods relies on using space small enough to fit in fast cache, and the Chernoff-style bounds [38, 41] all require that μ is much smaller than the space used. For larger values of μ, Pătraşcu and Thorup [38, 41] offered some weaker bounds with a deviation that was off by several logarithmic factors. It can be shown that some of these limitations are inherent to simple and twisted tabulation. For instance, they cannot even reliably distribute balls into $m = 2$ bins, as described in the first case above, if the expected number of balls in each bin exceeds the space used.

In this paper, we construct and analyse a new family of fast hash functions *tabulation-permutation* hashing that has Chernoff-style concentration bounds like (3) without any restrictions on μ. This generality is important if building a general online system with no knowledge of future input. Later, we shall give concrete examples from streaming where μ is in fact large. Our bounds hold for all of the cases described above and all possible inputs. Furthermore, tabulation-permutation hashing is an order of magnitude faster than any other known hash function with similar concentration bounds, and almost as fast as simple and twisted tabulation. We demonstrate this both theoretically and experimentally. Stepping back, our main theoretical contribution lies in the field of analysis of algorithms, and is in the spirit of Knuth's analysis of linear probing [29], which shows strong theoretical guarantees for a very practical algorithm. We show that tabulation-permutation hashing has strong theoretical Chernoff-style concentration bounds. Moreover, on the practical side, we perform experiments, summarized in Table 1, demonstrating that it is comparable in speed to some

of the fastest hash functions in use, none of which provide similar concentration bounds.

When talking about hashing in constant time, the actual size of the constant is of crucial importance. First, hash functions typically execute the same instructions on all keys, in which case we always incur the worst-case running time. Second, hashing is often an inner-loop bottle-neck of data processing. Third, hash functions are often applied in time-critical settings. Thus, even speedups by a multiplicative constant are very impactful. As an example from the Internet, suppose we want to process packets passing through a high-end Internet router. Each application only gets very limited time to look at the packet before it is forwarded. If it is not done in time, the information is lost. Since processors and routers use some of the same technology, we never expect to have more than a few instructions available. Slowing down the Internet is typically not an option. The papers of Krishnamurthy et al. [30] and Thorup and Zhang [46] explain in more detail how high speed hashing is necessary for their Internet traffic analysis. Incidentally, our hash function is a bit faster than the ones from [30, 46], which do not provide Chernoff-style concentration bounds.

Some concrete examples of applications of our new hash-family may be found in [1], where it is shown that certain classic streaming algorithms enjoy very substantial speed-ups when implemented using tabulation-permutation hashing; namely the original similarity estimation of Broder [9] and the estimation of distinct elements of Bar-Yossef et al. [6]. The strong concentration bounds makes the use of independent repetitions unnecessary, allowing the implementations of the algorithms to be both simpler and faster. We stress that in high-volume streaming algorithms, speed is of critical importance.

Tabulation-permutation hashing builds on top of simple tabulation hashing, and to analyse it, we require a new and better understanding of the behaviour and inherent limitations of simple tabulation, which we proceed to describe. Afterwards we break these limitations by introducing our new powerful tabulation-permutation hashing scheme.

We omit almost every proof, but they can be found in the full version of this paper [2].

1.1 Simple Tabulation Hashing

Simple tabulation hashing dates back to Zobrist [48]. In simple tabulation hashing, we consider the key domain U to be of the form $U = \Sigma^c$ for some character alphabet Σ and $c = O(1)$, such that each key consists of c characters of Σ. Let $m = 2^\ell$ be given and identify $[m] = \{0, 1, \dots, m-1\}$ with $[2]^\ell$. A simple tabulation hash function, $h\colon \Sigma^c \to [m]$, is then defined as follows. For each $j \in \{1, \dots, c\}$ store a fully random character table $h_j\colon \Sigma \to [m]$ mapping characters of the alphabet Σ to ℓ-bit hash values. To evaluate h on a key $x = (x_1, \dots, x_c) \in \Sigma^c$, we compute $h(x) = h_1(x_1) \oplus \cdots \oplus h_c(x_c)$, where $\oplus$ denotes bitwise XOR – an extremely fast operation. With character tables in cache, this scheme is the fastest known 3-independent hashing scheme [38]. We will denote by $u = |U|$ the size of the key domain, identify $U = \Sigma^c$ with $[u]$, and always assume the size of the alphabet, $|\Sigma|$, to be a power of two. For instance, we could consider 32-bit keys consisting of four 8-bit characters. For a given computer, the best choice of c in terms of speed is easily determined experimentally once and for all, and is independent of the problems considered.

Let $S \subseteq U$ and consider hashing $n = |S|$ weighted balls or keys into $m = 2^\ell$ bins using a simple tabulation function, $h\colon [u] \to [m]$, in line with the first case mentioned above. We prove the following theorem.

Theorem 1. *Let $h\colon [u] \to [m]$ be a simple tabulation hash function with $[u] = \Sigma^c$, $c = O(1)$. Let $S \subseteq [u]$ be given of size $n = |S|$ and assign to each key/ball $x \in S$ a weight $w_x \in [0, 1]$. Let $y \in [m]$, and define $X = \sum_{x \in S} w_x \cdot [h(x) = y]$ to be the total weight of the balls hashing to bin y. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability n/m^γ, where the constants of the asymptotics are determined solely by c and γ. Furthermore, this concentration is query invariant.*

In Theorem 1, we note that the expectation, $\mu = \mathbb{E}[X]$, and the variance, $\sigma^2 = \mathrm{Var}[X]$, are the same as if h were a fully random hash function since h is 3-independent. This is true even when conditioning on the hash value of a query key having a specific value. The bound provided by Theorem 1 is therefore the same as the variance based Chernoff bound (3) except for a constant delay in the exponential decrease and an added error probability of n/m^γ. Since $\sigma^2 \le \mu$, Theorem 1 also implies the classic one-sided Chernoff bounds (1) and (2), again with the constant delay and the added error probability as above, and a leading factor of 2.

Pătraşcu and Thorup [38] proved an equivalent probability bound, but without weights, and, more importantly, with the restriction that the number of bins $m \ge n^{1-1/(2c)}$. In particular, this implies the restriction $\mu \le |\Sigma|^{1/2}$. Our new bound gives Chernoff-style concentration with high probability in n for any $m \ge n^\varepsilon$, $\varepsilon = \Omega(1)$. Indeed, letting $\gamma' = (\gamma + 1)/\varepsilon$, the added error probability becomes $n/m^{\gamma'} \le 1/n^\gamma$.

However, for small m the error probability n/m^γ is prohibitive. For instance, unbiased coin tossing, corresponding to the case $m = 2$, has an added error probability of $n/2^\gamma$ which is useless. We will show that it is inherently impossible to get good concentration bounds using simple tabulation hashing when the number of bins m is small. To handle all instances, including those with few bins, and to support much more general Chernoff bounds, we introduce a new hash function: tabulation-permutation hashing.

1.2 Tabulation-Permutation Hashing

We start by defining *tabulation-permutation hashing* from Σ^c to Σ^d with $c, d = O(1)$. A tabulation-permutation hash function $h\colon \Sigma^c \to \Sigma^d$ is given as a composition, $h = \tau \circ g$, of a simple tabulation hash function $g\colon \Sigma^c \to \Sigma^d$ and a permutation $\tau\colon \Sigma^d \to \Sigma^d$. The permutation is a coordinate-wise fully random permutation: for each $j \in \{1, \dots, d\}$, pick a uniformly random character permutation $\tau_j : \Sigma \to \Sigma$. Now, $\tau = (\tau_1, \dots, \tau_d)$ in the sense that for $z = (z_1, \dots, z_d) \in \Sigma^d$, $\tau(z) = (\tau_1(z_1), \dots, \tau_d(z_d))$. In words, a tabulation-permutation hash function hashes c characters to d characters using simple tabulation, and then randomly permutes each of the d output characters. As is, tabulation-permutation hash functions yield values in Σ^d, but we will soon see how we can hash to $[m]$ for any $m \in \mathbb{N}$.

If we precompute tables $T_i : \Sigma \to \Sigma^d$, where

$$T_i(z_i) = \left(\overbrace{0,\ldots,0}^{i-1}, \tau_i(z_i), \overbrace{0,\ldots,0}^{d-i}\right), \quad z_i \in \Sigma,$$

then $\tau(z_1,\ldots,z_d) = T_1(z_1) \oplus \cdots \oplus T_d(z_d)$. Thus, τ admits the same implementation as simple tabulation, but with a special distribution on the character tables. If in particular $d \le c$, the permutation step can be executed at least as fast as the simple tabulation step.

Our main result is that with tabulation-permutation hashing, we get high probability Chernoff-style bounds for the third and most general case described in the beginning of the introduction.

Theorem 2. *Let $h : [u] \to [r]$ be a tabulation-permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Let $v : [u] \times [r] \to [0,1]$ be a fixed value function that to each key $x \in [u]$ assigns a value $X_x = v(x, h(x)) \in [0,1]$ depending on the hash value $h(x)$ and define $X = \sum_{x\in[u]} X_x$. For any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c, d, and γ. Furthermore, this concentration is query invariant.*

Tabulation-permutation hashing inherits the 3-independence of simple tabulation, so as in Theorem 1, $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$ have exactly the same values as if h were a fully-random hash function. Again, this is true even when conditioning on the hash value of a query key having a specific value.

Tabulation-permutation hashing allows us to hash into m bins for any $m \in \mathbb{N}$ (not necessarily a power of two) preserving the strong concentration from Theorem 2. To do so, simply define the hash function $h^m : [u] \to [m]$ by $h^m(x) = h(x) \bmod m$. Relating back to Theorem 1, consider a set $S \subseteq U$ of n balls where each ball $x \in S$ has a weight $w_x \in [0,1]$ and balls x outside S are defined to have weight $w_x = 0$. To measure the total weight of the balls landing in a given bin $y \in [m]$, we define the value function $v(x,z) = w_x \cdot [z \bmod m = y]$. Then

$$X = \sum_{x\in[u]} v(x,h(x)) = \sum_{x\in S} w_x \cdot [h^m(x) = y]$$

is exactly the desired quantity and we get the concentration bound from Theorem 2. Then the big advantage of tabulation-permutation hashing over simple tabulation hashing is that it reduces the added error probability from n/m^γ of Theorem 1 to the $1/u^\gamma$ of Theorem 2, where u is the size of the key universe. Thus, with tabulation-permutation hashing, we actually get Chernoff bounds with high probability regardless of the number of bins.

Pătraşcu and Thorup [41] introduced twisted tabulation that like our tabulation-permutation achieved Chernoff-style concentration bounds with a general value function v. Their bounds are equivalent to those of Theorem 2, but only under the restriction $\mu \le |\Sigma|^{1-\Omega(1)}$. To understand how serious this restriction is, consider again tossing an unbiased coin for each key x in a set $S \subseteq [u]$, corresponding to the case $m = 2$ and $\mu = |S|/2$. With the restriction from [41], we can only handle $|S| \le 2\,|\Sigma|^{1-\Omega(1)}$, but recall that Σ is chosen small enough for character tables to fit in fast cache, so this rules out any moderately large data set. We are going to show that for certain sets S, twisted tabulation has the same problems as simple tabulation when hashing to few bins. This implies that the restrictions from [41] cannot be lifted with a better analysis.

Pătraşcu and Thorup [41] were acutely aware of how prohibitive the restriction $\mu \le |\Sigma|^{1-\Omega(1)}$ is. For unbounded μ, they proved a weaker bound; namely that with twisted tabulation hashing, $X = \mu \pm O(\sigma(\log u)^{c+1})$ with probability $1 - u^{-\gamma}$ for any $\gamma = O(1)$. With our tabulation-permutation hashing, we get $X = \mu \pm O(\sigma(\log u)^{1/2})$ with the same high probability, $1 - u^{-\gamma}$. Within a constant factor on the deviation, our high probability bound is as good as with fully-random hashing.

More related work, including Siegel's [43] and Thorup's [44] highly independent hashing will be discussed in Section 1.7.

1.3 Tabulation-1Permutation

Above we introduced tabulation-permutation hashing which yields Chernoff-style bounds with an arbitrary value function. This is the same general scenario as was studied for twisted tabulation in [41]. However, for almost all applications we are aware of, we only need the generality of the second case presented at the beginning of the introduction. Recall that in this case we are only interested in the total weight of the balls hashing to a certain interval. As it turns out, a significant simplification of tabulation-permutation hashing suffices to achieve strong concentration bounds. We call this simplification *tabulation-1permutation*. Tabulation-permutation hashing randomly permutes each of the d output characters of a simple tabulation function $g : \Sigma^c \to \Sigma^d$. Instead, tabulation-1permutation only permutes the most significant character.

More precisely, a tabulation-1permutation hash function $h : \Sigma^c \to \Sigma^d$ is a composition, $h = \tau \circ g$, of a simple tabulation function, $g : \Sigma^c \to \Sigma^d$, and a random permutation, $\tau : \Sigma^d \to \Sigma^d$, of the most significant character, $\tau(z_1,\ldots,z_d) = (\tau_1(z_1), z_2, \ldots, z_d)$ for a random character permutation $\tau_1 : \Sigma \to \Sigma$.

To simplify the implementation of the hash function and speed up its evaluation, we can precompute a table $T : \Sigma \to \Sigma^d$ such that for $z_1 \in \Sigma$,

$$T(z_1) = \left(z_1 \oplus \tau_1(z_1), \overbrace{0,\ldots,0}^{d-1}\right).$$

Then if $g(x) = z = (z_1,\ldots,z_d)$, $h(x) = z \oplus T(z_1)$.

This simplified scheme, needing only $c+1$ character lookups, is powerful enough for concentration within an arbitrary interval.

Theorem 3. *Let $h : [u] \to [r]$ be a tabulation-1permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Consider a key/ball set $S \subseteq [u]$ of size $n = |S|$ where each ball $x \in S$ is assigned a weight $w_x \in [0,1]$. Choose arbitrary hash values $y_1, y_2 \in [r]$ with $y_1 \le y_2$. Define $X = \sum_{x\in S} w_x \cdot [y_1 \le h(x) < y_2]$ to be the total weight of balls hashing to the interval $[y_1, y_2)$. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c, d, and γ. Furthermore, this concentration is query invariant.*

One application of Theorem 3 is in the following sampling scenario: We set $y_1 = 0$, and sample all keys with $h(x) < y_2$. Each key is then sampled with probability y_2/r, and Theorem 3 gives concentration on the number of samples. In [1] this is used for more efficient implementations of streaming algorithms.

Another application is efficiently hashing into an arbitrary number $m \leq r$ of bins. We previously discussed using hash values modulo m, but a general mod-operation is often quite slow. Instead we can think of hash values as fractions $h(x)/r \in [0,1)$. Multiplying by m, we get a value in $[0,m)$, and the bin index is then obtained by rounding down to the nearest integer. This implementation is very efficient because r is a power of two, $r = 2^b$, so the rounding is obtained by a right-shift by b bits. To hash a key x to $[m]$, we simply compute $h^m(x) = (h(x) * m) >> b$. Then x hashes to bin $d \in [m]$ if and only if $d \in [y_1, y_2) \subseteq [r]$ where $y_1 = \lfloor rd/m \rfloor$ and $y_2 = \lfloor r(d+1)/m \rfloor$, so the number of keys hashing to a bin is concentrated as in Theorem 3. Moreover, h^m uses only $c+1$ character lookups and a single multiplication in addition to some very fast shifts and bit-wise Boolean operations.

1.4 Subpolynomial Error Probabilities

In Theorem 2 and 3, we have $\Pr[|X-\mu| \geq t] = O(\exp(-\Omega(\sigma^2 C(t/\sigma^2)))) + 1/u^\gamma$ which holds for any fixed γ. The value of γ affects the constant hidden in the Ω-notation delaying the exponential decrease. In the full version [2], we show that the same bound does not hold if γ is replaced by any slow-growing but unbounded function. Nevertheless, it follows from our analysis that for every $\alpha(u) = \omega(1)$ there exists $\beta(u) = \omega(1)$ such that whenever $\exp(-\sigma^2 C(t/\sigma^2)) < 1/u^{\alpha(u)}$, $\Pr[|X-\mu| \geq t] \leq 1/u^{\beta(u)}$.

1.5 Generic Remarks on Universe Reduction and Amount of Randomness

The following observations are fairly standard in the literature. Suppose we wish to hash a set of keys S belonging to some universe $\mathcal{U}$. The universe may be so large compared to S that it is not efficient to directly implement a theoretically powerful hashing scheme like tabulation-permutation hashing. A standard first step is to perform a *universe reduction*, mapping $\mathcal{U}$ randomly to "signatures" in $[u] = \{0,1,\ldots,u-1\}$, where $u = n^{O(1)}$, e.g. $u = n^3$, so that no two keys from S are expected to get the same signature [10]. As the only theoretical property required for the universe reduction is a low collision probability, this step can be implemented using very simple hash functions as described in [45]. In this paper, we generally assume that this universe reduction has already been done, if needed, hence that we only need to deal with keys from a universe $[u]$ of size polynomial in n. For any small constant $\varepsilon > 0$ we may thus pick $c = O(1/\varepsilon)$ such that the space used for our hash tables, $\Theta(|\Sigma|)$, is $O(n^\varepsilon)$. Practically speaking, this justifies focusing on the hashing of 32- and 64-bit keys.

When we defined simple tabulation above, we said the character tables were fully random. However, for the all the bounds in this paper, it would suffice if they were populated with a $O(\log u)$-independent pseudo-random number generator (PNG), so we only need a seed of $O(\log u)$ random words to be shared among all applications who want to use the same simple tabulation hash function. Then, as a preprocesing for fast hashing, each application can locally fill the character tables in $O(|\Sigma|)$ time [14]. Likewise, for our tabulation permutation hashing, our bounds only require a $O(\log u)$-independent PNG to generate the permutations. The basic point here is that tabulation based hashing does not need a lot of randomness to fill the tables, but only space to store the tables as needed for the fast computation of hash values.

1.6 Techniques

The paper relies on three main technical insights to establish the concentration inequality for tabulation-permutation hashing of Theorem 2. We shall here describe each of these ideas and argue that each is in fact necessary towards an efficient hash function with strong concentration bounds.

1.6.1 Improved Analysis of Simple Tabulation. The first step towards proving Theorem 2 is to better understand the distribution of simple tabulation hashing. We describe below how an extensive combinatorial analysis makes it possible to prove a generalised version of Theorem 1.

To describe the main idea of this technical contribution, we must first introduce some ideas from previous work in the area. This will also serve to highlight the inherent limitations of previous approaches. A simplified account is the following. Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function, let $y \in [m]$ be given, and for some subset of keys $S \subseteq \Sigma^c$, let $X = \sum_{x \in S}[h(x) = y]$ be the random variable denoting the number of elements $x \in S$ that have hash value $h(x) = y$. Our goal is to bound the deviation of X from its mean $\mu = |S|/m$. We first note that picking a random simple tabulation hash function $h : \Sigma^c \to [m]$ amounts to filling the c character tables, each of size Σ, with uniformly random hash values. Thus, picking a simple tabulation hash function $h : \Sigma^c \to [m]$ corresponds to picking a uniformly random hash function $h\colon [c] \times \Sigma \to [m]$. We call $[c] \times \Sigma$ the set of *position characters*. Viewing a key $x = (x_1, \ldots, x_c) \in \Sigma^c$ as a set of position characters, $x = \{(1,x_1), \ldots, (c,x_c)\}$, and slightly abusing notation, it then holds that $h(x) = \bigoplus_{\alpha \in x} h(\alpha)$. Now let $\alpha_1, \ldots, \alpha_r$ be a (for the sake of the proof) well-chosen ordering of the position characters. For each $k \in [r+1]$, we define the random variable $X_k = \mathbb{E}[X \mid h(\alpha_1), \ldots, h(\alpha_k)]$, where $h(\alpha_i)$ is the value of the entry of the lookup table of h corresponding to α_i. The process $(X_k)_{k=0}^r$ is then a martingale. We can view this as revealing the lookup table of h one entry at a time and adjusting our expectation of the outcome of X accordingly. Defining the martingale difference $Y_k = X_k - X_{k-1}$, we can express X as a sum $X = \mu + \sum_{k=1}^{c\cdot|\Sigma|} Y_k$. Previous work has then bounded the sum using a Chernoff inequality for martingales as follows. Due to the nature of the ordering of $\{\alpha_i\}_{i=1}^r$, we can find $M > 0$ such that with high probability, $|Y_k| \leq M$ for every k. Then conditioned on each of the Y_ks being bounded, X satisfies the Chernoff bounds of (1) and (2) except the exponent is divided by M. As long as the expectation, μ, satisfies $\mu = O(|\Sigma|)$, it is possible[2] that $M = O(1)$, yielding Chernoff bounds with a constant in the delay of the exponential decrease. However, since there are only $c \cdot |\Sigma|$ variables, Y_k, it is clear that $M \geq \mu/(c \cdot |\Sigma|)$. Thus, whenever $\mu = \omega(|\Sigma|)$, the delay of the exponential decrease is super-constant, meaning that we do not get asymptotically tight Chernoff-style bounds. This obstacle has been an inherent issue with the previous techniques in analysing both simple tabulation [38] as well as twisted tabulation [41]. Being

[2] In [38], the actual analysis of simple tabulation using this approach achieves $\mu = O(\sqrt{|\Sigma|})$.

unable to bound anything beyond the absolute deviation of each variable Y_k, it is impossible to get good concentration bounds for large expectations, μ.

Going beyond the above limitation, we dispense with the idea of bounding absolute deviations and instead bound the sum of variances, $\sigma^2 = \sum_{k=1}^{c\cdot|\Sigma|} \mathrm{Var}[Y_k]$. This sum has a combinatorial interpretation relating to the number of collisions of hash keys, i.e., the number of pairs $y_1, y_2 \in \Sigma^c$ with $h(y_1) = h(y_2)$.

An extensive combinatorial analysis of simple tabulation hashing yields high-probability bounds on the sum of variances that is tight up to constant factors. This is key in establishing an induction that allows us to prove Theorem 1. Complementing our improved bounds, we will show that simple tabulation hashing inherently does not support Chernoff-style concentration bounds for small m.

1.6.2 Permuting the Hash Range. Our next step is to consider the hash function $h = \tau \circ g \colon \Sigma^c \to \Sigma$ where $g \colon \Sigma^c \to \Sigma$ is a simple tabulation hash function and $\tau \colon \Sigma \to \Sigma$ is a uniformly random permutation. Our goal is to show that h provides good concentration bounds for any possible value function. To showcase our approach, we consider the example of hashing to some small set, $[m]$, of bins, e.g., with $m = 2$ as in our coin tossing example. This can be done using the hash function $h^m \colon \Sigma^c \to [m]$ defined by $h^m(x) = (h(x) \bmod m)$. For simplicity we assume that m is a power of two, or equivalently, that m divides $|\Sigma|$. We note that the case of small m was exactly the case that could not be handled with simple tabulation hashing alone.

Let us look at the individual steps of h^m. First, we use simple tabulation mapping into the "character bins", Σ. The number of balls in any given character bin is nicely concentrated, but only because $|\Sigma|$ is large. Next, perform a permutation followed by the mod m operation. The last two steps correspond to the way we would deal a deck of $|\Sigma|$ cards into m hands. The cards are shuffled by a random permutation, then dealt to the m players one card at a time in cyclic order. The end result is that each of the final m bins is assigned exactly $|\Sigma|/m$ random character bins. An important point is now that because the partitioning is *exact*, the error in the number of balls in a final bin stems solely from the errors in the $|\Sigma|/m$ character bins, and because the partitioning is *random*, we expect the positive and negative errors to cancel out nicely. The analysis, which is far from trivial, requires much more than these properties. For example, we also need the bound described in Section 1.6.1 on the sum of variances. This bound ensures that not only is the number of balls in the individual character bins nicely concentrated around the mean, but moreover, there is only a small number of character bins for which the error is large. That these things combine to yield strong concentration, not only in the specific example above, but for general value functions as in Theorem 2, is quite magical.

We finish the discussion by mentioning two approaches that do not work and highlight how a permutation solves the issues of these strategies.

First, one may ask why we need the permutation at all. After all, the mod m operation also partitions the $|\Sigma|$ character bins into groups of the same size, $|\Sigma|/m$. The issue is that while a simple tabulation hash function, $g : \Sigma^c \to \Sigma$, has good concentration in each of the individual character bins, the $|\Sigma|/m$ character bins being picked out by the mod m operation constitute a very structured subset of Σ, and the errors from this set of bins could be highly correlated. Indeed, in the full version [2] we show both theoretically and experimentally that the structure of simple tabulation causes this to happen for certain sets of keys.

Second, the reader may wonder why we use a permutation, $\tau \colon \Sigma \to \Sigma$, instead of a random hash function as in double tabulation [44]. In terms of the card dealing analogy, this would correspond to throwing the $|\Sigma|$ cards at the m astonished card players one at a time with a random choice for each card, not guaranteeing that the players each get the same number of cards. And this is exactly the issue. Using a fully random hash function τ', we incur an extra error stemming from τ' distributing the $|\Sigma|$ character bins unevenly into the final bins. This is manifested in the variance of the number of balls hashing to a specific bin: Take again the coin tossing example with $n \geq |\Sigma|$ balls being distributed into $m = 2$ bins. With a permutation τ the hash function becomes 2-independent, so the variance is the same as in the fully random setting, $n/4$. Now even if g distributes the n keys into the character bins evenly, with exactly n/Σ keys in each, with a fully random hash function, τ', the variance becomes $(n/|\Sigma|)^2 \cdot |\Sigma|/4 = n^2/(4|\Sigma|)$, a factor of $n/|\Sigma|$ higher.

1.6.3 Squaring the Hash Range. The last piece of the puzzle is a trick to extend the range of a hash function satisfying Chernoff-style bounds. We wish to construct a hash function $h \colon \Sigma^c \to [m]$ satisfying Chernoff-style bounds for m arbitrarily large as in Theorem 2. At first sight, the trick of the previous subsection would appear to suffice for the purpose. However, if we let $g = \tau \circ h$ be the composition of a simple tabulation hash function $h \colon \Sigma^c \to [m]$ and τ a random permutation of $[m]$, we run into trouble if for instance $[m] = \Sigma^c$. In this case, a random permutation of $[m]$ would require space equal to that of a fully random function $f \colon \Sigma^c \to [m]$, but the whole point of hashing is to use less space. Hence, we instead prove the following. Let $a \colon C \to D$ and $b \colon C \to D$ be two independent hash functions satisfying Chernoff-style bounds for general value functions. Then this property is preserved up to constant factors under "concatenation", i.e., if we let $c \colon C \to D^2$ be given by $c(x) = (a(x), b(x))$, then c is also a hash function satisfying Chernoff-style bounds for general value functions, albeit with a slightly worse constant delay in the exponential decrease than a and b. Thus, this technique allows us to "square" the range of a hash function.

With this at hand, let $h_1, h_2 \colon \Sigma^c \to \Sigma$ be defined as $h_1 = \tau_1 \circ g_1$ and $h_2 = \tau_2 \circ g_2$, where $g_1, g_2 \colon \Sigma^c \to \Sigma$ are simple tabulation hash functions and $\tau_1, \tau_2 \colon \Sigma \to \Sigma$ are random permutations. Then the concatenation $h \colon \Sigma^c \to \Sigma^2$ of h_1 and h_2 can be considered a composition of a simple tabulation function $g \colon \Sigma^c \to \Sigma^2$ given by $g(x) = (g_1(x), g_2(x))$ and a coordinate-wise permutation $\tau = (\tau_1, \tau_2) \colon \Sigma^2 \to \Sigma^2$, where the latter is given by $\tau(x_1, x_2) = (\tau_1(x_1), \tau_2(x_2)), x_1, x_2 \in \Sigma$. Applying our composition result, gives that g also satisfies Chernoff-style bounds. Repeating this procedure $\lceil \log(d) \rceil = O(1)$ times, yields the desired concentration bound for tabulation-permutation hashing $h \colon \Sigma^c \to \Sigma^d$ described in Theorem 2.

1.7 Related Work – Theoretical and Experimental Comparisons

In this section, we shall compare the performance of tabulation-permutation and tabulation-1permutation hashing with other related results. Our comparisons are both theoretical and empirical. Our goal in this paper is fast constant-time hashing having strong concentration bounds with high probability, i.e., bounds of the form

$$\Pr[|X-\mu| \geq t] \leq 2\exp(-\Omega(\sigma^2 C(t/\sigma^2))) + u^{-\gamma},$$

as in Definition 1 and Theorems 2 and 3, or possibly with σ^2 replaced by $\mu \geq \sigma^2$. Theoretically, we will only compare with other hashing schemes that are relevant to this goal. In doing so, we distinguish between the hash functions that achieve Chernoff-style bounds with restrictions on the expected value and those that, like our new hash functions, do so without such restrictions, which is what we want for all possible input. Empirically, we shall compare the practical evaluation time of tabulation-permutation and permutation-1permutation to the fastest commonly used hash functions and to hash functions with similar theoretical guarantees. A major goal of algorithmic analysis is to understand the theoretical behavior of simple algorithms that work well in practice, providing them with good theoretical guarantees such as worst-case behavior. For instance, one may recall Knuth's analysis of linear probing [29], showing that this very practical algorithm has strong theoretical guarantees. In a similar vein, we not only show that the hashing schemes of tabulation-permutation and tabulation-1permutation have strong theoretical guarantees, we also perform experiments, summarized in Table 1, demonstrating that in practice they are comparable in speed to some of the most efficient hash functions in use, none of which have similar concentration guarantees. Thus, with our new hash functions, hashing with strong theoretical concentration guarantees is suddenly feasible for time-critical applications.

1.7.1 High Independence and Tabulation. Before this paper, the only known way to obtain unrestricted Chernoff-style concentration bounds with hash functions that can be evaluated in constant time was through k-independent hashing. Recall that a hash function $h : U \to R$ is k-independent if the distribution of $(h(x_1), \ldots, h(x_k))$ is uniform in R^k for every choice of distinct keys $x_1, \ldots, x_k \in U$. Schmidt, Siegel, and Srinivasan [42] have shown that with k-independent hashing, we have Chernoff-style concentration bounds in all three cases mentioned at the beginning of the introduction up to an added error probability decreasing exponentially in k. With $k = \Theta(\gamma \log u)$, this means Chernoff-style concentration with an added error probability of $1/u^\gamma$ like in Theorem 2 and 3. However, evaluating any k-independent hash function takes time $\Omega(k)$ unless we use a lot of space. Indeed, a cell probe lower bound by Siegel [43] states that evaluating a k-independent hash function over a key domain $[u]$ using $t < k$ probes, requires us to use at least $u^{1/t}$ cells to represent the hash function. Thus, aiming for Chernoff concentration through k-independence with $k = \Omega(\log u)$ and with constant evaluation time, we would have to use $u^{\Omega(1)}$ space like our tabulation-permutation. Here it should be mentioned that k-independent PolyHash modulo a prime p can be evaluated at k points in total time $O(k \log^2 k)$ using multipoint evaluation methods. Then the average evaluation time is $O(\log^2 k)$, but it requires that the hashing can be done to batches of k keys at a time. We can no longer hash one key at a time, continuing with other code before we hash the next key. This could be a problem for some applications. A bigger practical issue is that it is no longer a black box implementation of a hash function. To understand the issue, think of Google's codebase where thousands of programs are making library calls to hash functions. A change to multipoint evaluation would require rewriting all of the calling programs, checking in each case that batch hashing suffices — a huge task that likely would create many errors. A final point is that multipoint evaluation is complicated to implement yet still not as fast as our tabulation-permutation hashing.

Turning to upper bounds, Siegel designed a $u^{\Omega(1/c^2)}$-independent hash function that can be represented in tables of size $u^{1/c}$ and evaluated in $c^{O(c)}$ time. With $c = O(1)$, this suffices for Chernoff-style concentration bounds by the argument above. However, as Siegel states, the hashing scheme is "far too slow for any practical application".

In the setting of Siegel, Thorup's double tabulation [44] is a simpler and more efficient construction of highly independent hashing. It is the main constant-time competitor of our new tabulation-permutation hashing, and yet it is 30 times slower in our experiments. In the following, we describe the theoretical guarantees of double tabulation hashing and discuss its concrete parameters in terms of speed and use of space towards comparing it with tabulation-permutation hashing.

A *double tabulation* hash function, $h : \Sigma^c \to \Sigma^c$ is the composition of two independent simple tabulation hash functions $h_1 : \Sigma^c \to \Sigma^d$ and $h_2 : \Sigma^d \to \Sigma^c$, $h = h_2 \circ h_1$. Evaluating the function thus requires $c + d$ character lookups. Assuming that each memory unit stores an element from $[u] = \Sigma^c$ and $d \geq c$, the space used for the character tables is $(c(d/c) + d)u^{1/c} = 2du^{1/c}$. Thorup [44] has shown that if $d \geq 6c$, then with probability $1 - o(\Sigma^{2-d/(2c)})$ over the choice of h_1, the double tabulation hash function h is k-independent for $k = |\Sigma|^{1/(5c)} = u^{\Omega(1/c^2)}$. More precisely, with this probability, the output keys $(h_1(x))_{x\in\Sigma^c}$ are distinct, and h_2 is k-independent when restricted to this set of keys. If we are lucky to pick such an h_1, this means that we get the same high indepence as Siegel [43]. With $d = 6c$, the space used is $12cu^{1/c} = O(cu^{1/c})$ and the number of character lookups to compute a hash value is $7c = O(c)$. Tabulation-permutation hashing is very comparable to Thorup's double tabulation. As previously noted, it can be implemented in the same way, except that we fill the character tables of h_2 with permutations and padded zeros instead of random hash values. To compare, a tabulation-permutation hash function $h\colon \Sigma^c \to \Sigma^c$ requires $2c$ lookups and uses space $2cu^{1/c}$, which may not seem a big difference. However, in the following, we demonstrate how restrictions on double tabulation cost an order of magnitude in speed and space compared with tabulation-permutation hashing when used with any realistic parameters.

With Thorup's double tabulation, for $(\log u)$-independence, we need $\log u \leq |\Sigma|^{1/(5c)} = u^{1/(5c^2)}$. In choosing values for u and c that work in practice, this inequality is very restrictive. Indeed, even for $c = 2$, $\log u \leq u^{1/20}$, which roughly implies that $\log u \geq 140$. Combined with the fact that the character tables use space $12c|\Sigma|$, and that $|\Sigma| \geq (\log u)^{5c}$, this is an intimidating amount of space. Another problem is the error probability over h_1 of $1 - o(\Sigma^{2-d/(2c)})$.

Table 1: The time for different hash functions to hash 10^7 keys of length 32 bits and 64 bits, respectively, to ranges of size 32 bits and 64 bits. The experiment was carried out on two computers. The hash functions written in italics are those without general Chernoff-style bounds. Hash functions written in bold are the contributions of this paper. The hash functions in regular font are known to provide Chernoff-style bounds. Note that we were unable to implement double tabulation from 64 bits to 64 bits since the hash tables were too large to fit in memory.

	Running time (ms)			
	Computer 1		Computer 2	
Hash function	32 bits	64 bits	32 bits	64 bits
Multiply-Shift	4.2	7.5	23.0	36.5
2-Independent PolyHash	14.8	20.0	72.2	107.3
Simple Tabulation	13.7	17.8	53.1	55.9
Twisted Tabulation	17.2	26.1	65.6	92.5
Mixed Tabulation	28.6	68.1	120.1	236.6
Tabulation-1Permutation	16.0	19.3	63.8	67.7
Tabulation-Permutation	27.3	43.2	118.1	123.6
Double Tabulation	1130.1	–	3704.1	–
"Random" (100-Independent PolyHash)	2436.9	3356.8	7416.8	11352.6

Table 2: Theoretical time and space consumption of some of the hash functions discussed.

Hash function	Time	Space	Concentration Guarantee	Restriction
Multiply-Shift	$O(1)$	$O(1)$	Chebyshev's inequality	None
k-Independent PolyHash	$O(k)$	$O(k)$	Chernoff-style bounds	Requires $k = \Omega(\log u)$ for added error probability $O(1/u^\gamma)$
Simple Tabulation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Added error probability: $O(n/m^\gamma)$
Twisted Tabulation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Requires: $\mu \leq \lvert\Sigma\rvert^{1-\Omega(1)}$
Mixed Tabulation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Requires: $\mu = o(\lvert\Sigma\rvert)$
Tabulation-Permutation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Added error probability: $O(1/u^\gamma)$
Double Tabulation	$O(c^2)$	$O(u^{1/c})$	Chernoff-style bounds	Added error probability: $O(1/u^\gamma)$

If we want this to be $O(1/u)$, like in the error bounds from Theorem 2 and 3, we need $d \geq 2(c^2 + 2c)$. Thus, while things work well asymptotically, these constraints make it hard to implement highly independent double tabulation on any normal computer. However, based on a more careful analysis of the case with 32-bit keys, Thorup shows that using $c = 2$ characters of 16 bits, and $d = 20$ derived characters, gives a 100-independent hash function with probability $1 - 1.5 \times 10^{-42}$. According to [44] we cannot use significantly fewer resources even if we just want 4-independence. For hashing 32-bit keys, this means making 22 lookups for each query and using tables of total size $40 \cdot 2^{16}$. In contrast, if we hash 32-bit keys with tabulation-permutation hashing, we may use 8-bit characters with $d = c = 4$, thus making only 8 lookups in tables of total size $8 \cdot 2^8$. For this setting of parameters, our experiments (summarized in Table 1) show that double tabulation is approximately 30 times slower than tabulation-permutation hashing. For 64-bit keys, Thorup [44] suggests implementing double tabulation with $c = 3$ characters of 22 bits and $d = 24$. This would require 26 lookups in tables of total size $48 \cdot 2^{22}$. We were not able to implement this on a regular laptop due to the space requirement.

We finally mention that Christani et al. [15] have presented a hash family which obtains the even higher independence $u^{\Omega(1/c)}$. The scheme is, however, more complicated with a slower evaluation time of $\Theta(c \log c)$.

1.7.2 Space Bounded Independence and Chernoff Bounds. One of the earliest attempts of obtaining strong concentration bounds via hashing is a simple and elegant construction by Dietzfelbinger and Meyer auf der Heide [20]. For some parameters m, s, d, their hash family maps to $[m]$, can be represented with $O(s)$ space, and uses a $(d+1)$-independent hash function as a subroutine, where $d = O(1)$, e.g., a degree-d polynomial. In terms of our main goal of Chernoff-style bounds, their result can be cast as follows: Considering the number of balls from a fixed, but unknown, subset $S \subseteq U$, with $|S| = n$, that hashes to a specific bin, their result yields Chernoff bounds like ours with a constant delay in the exponential decrease and with an added error probability of $n\left(\frac{n}{ms}\right)^d$. The expected number of balls in a given bin is $\mu = n/m$, so the added error probability is $n(\mu/s)^d$. To compare with tabulation-permutation, suppose we insist on using space $O(|\Sigma|)$ and that we moreover want the added error probability to be $u^{-\gamma} = |\Sigma|^{-c\gamma}$ like in Theorems 2 and 3. With the hashing scheme from [20], we then need $\mu = O(|\Sigma|^{1-\gamma c/d})$. If we want to be able to handle expectations of order, e.g. $|\Sigma|^{1/2}$, we thus need $d \geq 2c\gamma$. For 64-bit key, $c = 8$, and $\gamma = 1$, say, this means that we need to evaluate a 16-independent hash function. In general, we see that the concentration bound above suffers from the same issues as those provided by Pătraşcu and Thorup for simple

and twisted tabulation [38, 41], namely that we only have Chernoff-style concentration if the expected value is much smaller than the space used.

Going in a different direction, Dietzfelbinger and Rink [21] use universe splitting to create a hash function that is highly independent (building on previous works [22, 23, 25, 27]) but, contrasting double tabulation as described above, only within a fixed set S, not the entire universe. The construction requires an upper bound n on the size of S, and a polynomial error probability of $n^{-\gamma}$ is tolerated. Here $\gamma = O(1)$ is part of the construction and affects the evaluation time. Assuming no such error has occurred, which is not checked, the hash function is highly independent on S. As with Siegel's and Thorup's highly independent hashing discussed above, this implies Chernoff bounds without the constant delay in the exponential decrease, but this time only within the fixed set S. In the same setting, Pagh and Pagh [37] have presented a hash function that uses $(1 + o(1))n$ space and which is fully independent on any given set S of size at most n with high probability. This result is very useful, e.g., as part of solving a static problem of size n using linear space, since, with high probability, we may assume fully-random hashing as a subroutine. However, from a Chernoff bound perspective, the fixed polynomial error probability implies that we do not benefit from any independence above $O(\log n)$, using the aforementioned results from [42]. More importantly, we do not want to impose any limitations to the size of the sets we wish to hash in this paper. Consider for example the classic problem of counting distinct elements in a huge data stream. The size of the data stream might be very large, but regardless, the hashing schemes of this paper will only use space $O(u^{1/c})$ with c chosen small enough for hash tables to fit in fast cache.

Finally, Dahlgaard et al. [17] have shown that on a given set S of size $|S| \leq |\Sigma|/2$ a double tabulation hash function, $h = h_2 \circ h_1$ as described above, is fully random with probability $1 - |\Sigma|^{1-\lfloor d/2 \rfloor}$ over the choice of h_1. For an error probability of $1/u$, we set $d = (2c + 2)$ yielding a hash function that can be evaluated with $3c + 2$ character lookups and using $(4c + 4)|\Sigma|$ space. This can be used to simplify the above construction by Pagh and Pagh [37]. Dahlgaard et al. [17] also propose mixed tabulation hashing which they use for statistics over k-partitions. Their analysis is easily modified to yield Chernoff-style bounds for intervals similar to our bounds for tabulation-1permuation hashing presented in Theorem 3, but with the restriction that the expectation μ is at most $|\Sigma|/\log^{2c}|\Sigma|$. This restriction is better than the earlier mentioned restictions $\mu \leq |\Sigma|^{1/2}$ for simple tabulation [38] and $\mu \leq |\Sigma|^{1-\Omega(1)}$ for twisted tabulation [41]. For mixed tabulation hashing, Dahlgaard et al. use $3c + 2$ lookups and $(5c + 4)|\Sigma|$ space. In comparison, tabulation-1permutation hashing, which has no restriction on μ, uses only $c + 1$ lookups and $(c + 1)|\Sigma|$ space.

1.7.3 Small Space Alternatives in Superconstant Time. Finally, there have been various interesting developments regarding hash functions with small representation space that, for example, can hash n balls to n bins such that the maximal number of balls in any bin is $O(\log n/\log\log n)$, corresponding to a classic Chernoff bound. Accomplishing this through independence of the hash function, this requires $O(\log n/\log\log n)$-independence and evaluation time unless we switch to hash functions using a lot of space as described above. However, [11, 33] construct hash families taking a random seed of $O(\log\log n)$ words and which can be evaluated using $O((\log\log n)^2)$ operations, still obtaining a maximal load in any bin of $O(\log n/\log\log n)$ with high probability. This is impressive as it only uses a small amount of space and a short random seed, though it does require some slightly non-standard operations when evaluating the hash functions. The running time however, is not constant, which is what we aim for in this paper.

A different result is by [26] who construct hash families which hash n balls to 2 bins. They construct hash families that taking a random seed of $O((\log\log n)^2)$ words get Chernoff bounds with an added error probability of $n^{-\gamma}$ for some constant γ, which is similar to our bounds. Nothing is said about the running time of the hash function of [26]. Since one of our primary goals is to design hash functions with constant running time, this makes the two results somewhat incomparable.

1.7.4 Experiments and Comparisons. To better understand the real-world performance of our new hash functions in comparison with well-known and comparable alternatives, we performed some simple experiments on regular laptops, as presented in Table 1. We did two types of experiments.

- On the one hand we compared with potentially faster hash functions with weaker or restricted concentration bounds to see how much we lose in speed with our theoretically strong tabulation-permutation hashing. We shall see that our tabulation-permutation is very competitive in speed.
- On the other hand we compared with the fastest previously known hashing schemes with strong concentration bounds like ours. Here we will see that we gain a factor of 30 in speed.

Concerning weaker, but potentially faster, hashing schemes we have chosen two types of hash functions for the comparison. First, we have the fast 2-independent hash functions multiply-shift (with addition) and 2-independent PolyHash. They are among the fastest hash functions in use and are commonly used in streaming algorithms. It should be noted that when we use 2-independent hash functions, the variance is the same as with full randomness, and it may hence suffice for applications with constant error probability. Furthermore, for data sets with sufficient entropy, Chung, Mitzenmacher, and Vadhan [16] show that 2-independent hashing suffices. However, as previously mentioned, we want provable Chernoff-style concentration bounds of our hash functions, equivalent up to constant factors to the behavior of a fully random hash function, for any possible input. Second, we have simple tabulation, twisted tabulation, and mixed tabulation, which are tabulation based hashing schemes similar to tabulation-1permutation and tabulation-permutation hashing, but with only restricted concentration bounds. It is worth noting that Dahlgaard, Knudsen, and Thorup [18] performed experiments showing that the popular hash functions MurmurHash3 [4] and CityHash [40] along with the cryptographic hash function Blake2 [5] all are slower than mixed tabulation hashing, which we shall see is even slower than permutation-tabulation hashing. These hash functions are used in practice, but given that our experiments show mixed tabulation to be slightly slower than tabulation-permutation hashing, these

can now be replaced with our faster alternatives that additionally provide theoretical guarantees as to their effectiveness.

Concerning hashing schemes with previous known strong concentration bounds, we compared with double tabulation and 100-independent PolyHash, which are the strongest competitors that we are aware of using simple portable code.

The experiment measures the time taken by various hash functions to hash a large set of keys. Since the hash functions considered all run the same instructions for all keys, the worst- and best-case running times are the same, and hence choosing random input keys suffices for timing purposes. Further technical details of the experiments are covered in the full version [2]. We considered both hashing 32-bit keys to 32-bit hash values and 64-bit keys to 64-bits hash values. We did not consider larger key domains as we assume that a universe reduction, as described in Section 1.5, has been made if needed. The results are presented in Table 1. Below, we comment on the outcome of the experiment for each scheme considered.

Multiply-Shift. The fastest scheme of the comparison is Dietzfelbinger's 2-independent Multiply-Shift [19]. For 32-bit keys it uses one 64-bit multiplication and a shift. For 64-bit keys it uses one 128-bit multiplication and a shift. As expected, this very simple hash function was the fastest in the experiment.

2-Independent PolyHash. We compare twice with the classic k-independent PolyHash [47]. Once with $k = 2$ and again with $k = 100$. k-independent PolyHash is based on evaluating a random degree $(k-1)$-polynomial over a prime field, using Mersenne primes to make it fast: $2^{61}-1$ for 32-bit keys and $2^{89}-1$ for 64-bit keys. The 2-independent version was 2-3 times slower in experiments than multiply-shift. It is possible that implementing PolyHash with a specialized carry-less multiplication [31] would provide some speedup. However, we do not expect it to become faster than multiply-shift.

Simple Tabulation. The baseline for comparison of our tabulation-based schemes is simple tabulation hashing. Recall that we hash using c characters from $\Sigma = [u^{1/c}]$ (in this experiment we considered $u = 2^{32}$ and $u = 2^{64}$). This implies c lookups from the character tables, which have total size $c\,|\Sigma|$. For each lookup, we carry out a few simple AC^0 operations, extracting the characters for the lookup and applying an XOR. Since the size of the character alphabet influences the lookup times, it is not immediately clear, which choice of c will be the fastest in practice. This is, however, easily checkable on any computer by simple experiments. In our case, both computers were fastest with 8-bit characters, hence with all character tables fitting in fast cache.

Theoretically, tabulation-based hashing methods are incomparable in speed to multiply-shift and 2-independent PolyHash, since the latter methods use constant space but multiplication which has circuit complexity $\Theta(\log w/\log\log w)$ for w-bit words [12]. Our tabulation-based schemes use only AC^0 operations, but larger space. This is an inherent difference, as 2-independence is only possible with AC^0 operations using a large amount of space [3, 32, 34]. As is evident from Table 1, our experiments show that simple tabulation is 2-3 slower than multiply-shift, but as fast or faster than 2-independent PolyHash. Essentially, this can be ascribed to the cache of the two computers used being comparable in speed to arithmetic instructions. This is not surprising as most computation in the world involves data and hence cache. It is therefore expected that most computers have cache as fast as arithmetic instructions. In fact, since fast multiplication circuits are complex and expensive, and a lot of data processing does not involve multiplication, one could imagine computers with much faster cache than multiplication [28].

Twisted Tabulation. Carrying out a bit more work than simple tabulation, twisted tabulation performs c lookups of entries that are twice the size, as well as executing a few extra AC^0 operations. It hence performs a little worse than simple tabulation hashing.

Mixed Tabulation. We implemented mixed tabulation hashing with the same parameters ($c = d$) as in [18]. With these parameters the scheme uses $2c$ lookups from $2c$ character tables, where c of the lookups are to table entries that are double as long as the output, which may explain its worse performance with 64-bit domains. In our experiments, mixed tabulation performs slightly worse than tabulation-permutation hashing. Recall from above that mixed tabulation is faster than many popular hash functions without theoretical guarantees, hence so is our tabulation-permutation.

Tabulation-1Permutation. Also only slightly more involved than simple tabulation, tabulation-1permutation performs $c + 1$ lookups using $c + 1$ character tables. In our experiments, tabulation-1permutation turns out to be a little bit faster than twisted tabulation, at most 30% slower than simple tabulation, and at most 4 times slower than multiply-shift. Recall that tabulation-1permutation is our hash function of choice for streaming applications where speed is critical.

Tabulation-Permutation. Tabulation-permutation hashing performs $2c$ lookups from $2c$ character tables. In our experiments, it is slightly more than twice as slow as simple tabulation, and at most 8 times slower than multiply-shift. It is also worth noting that it performs better than mixed tabulation.

Double Tabulation. Recall that among the schemes discussed so far, only tabulation-permutation and tabulation-1permutation hashing offer unrestricted Chernoff-style concentration with high probability. Double tabulation is the first alternative with similar guarantees and in our experiments it is 30 times slower for 32-bit keys. For 64-bit keys, we were unable to run it on the computers at our disposal due to the large amount of space required for the hash tables. As already discussed, theoretically, double tabulation needs more space and lookups. The 32-bit version performed 26 lookups in tables of total size $48 \cdot 2^{22}$, while tabulation-permutation only needs 8 lookups using $8 \cdot 2^8$ space. It is not surprising that double tabulation lags so far behind.

100-Independent PolyHash. Running the experiment with 100-independent PolyHash, it turned out that for 32-bit keys, it is slower than 100-independent double tabulation. A bit surprisingly, 100-independent PolyHash ran nearly 200 times slower than the 2-independent PolyHash, even though it essentially just runs the same code 99 times. An explanation could be that the 2-independent scheme just keeps two coefficients in registers while the 100-independent scheme would loop through all the coefficients. We remark that the number 100 is somewhat arbitrary. We need

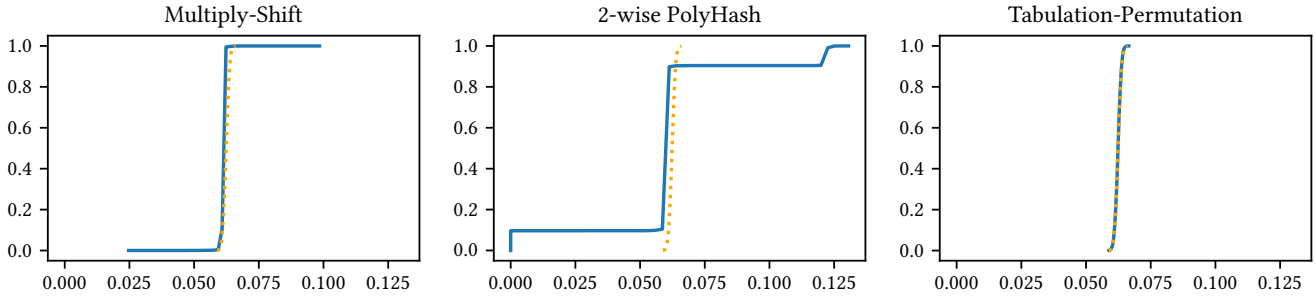

Figure 1: Cumulative distribution of the fraction of keys landing in a specific bin when hashing the arithmetic progression $\{a \cdot i \mid i \in [50000]\}$ to 16 bins for a random integer a. The dotted line is a 100-independent PolyHash.

$k = \Theta(\log u)$, but we do not know the exact constants in the Chernoff bounds with k-independent hashing. The running times are, however, easily scalable and for k-independent PolyHash, we would expect the evaluation time to change by a factor of roughly $k/100$.

Bad instances for Multiply-Shift and 2-wise PolyHash. We finally present experiments demonstrating concrete bad instances for the hash functions Multiply-Shift [19] and 2-wise PolyHash, underscoring what it means for them to not support Chernoff-style concentration bounds. In each case, we compare with our new tabulation-permutation hash function as well as 100-independent PolyHash, which is our approximation to an ideal fully random hash function. We refer the reader to [2] for bad instances for simple-tabulation [48] and twisted tabulation [41] as well as a more thorough discussion of our experiments.

Bad instances for Multiply-Shift and 2-independent PolyHash are analyzed in detail in [39, Appendix B]. The specific instance we consider is that of hashing the arithmetic progression $A = \{a \cdot i \mid i \in [50000]\}$ into 16 bins, where we are interested in the number of keys from A that hashes to a specific bin. We performed this experiment 5000 times, with independently chosen hash functions. The cumulative distribution functions on the number of keys from A hashing to a specific bin is presented in Figure 1. We see that most of the time 2-independent PolyHash and Multiply-Shift distribute the keys perfectly with exactly 1/16 of the keys in the bin. By 2-independence, the variance is the same as with fully random hashing, and this should suggest a much heavier tail, which is indeed what our experiments show. For contrast, we see that the cumulative distribution function with our tabulation-permutation hash function is almost indistinguishable from that of 100-independent Poly-Hash. We note that no amount of experimentation can prove that tabulation-permutation (or any other hash function) works well for all possible inputs. However, given the mathematical concentration guarantee of Theorem 2, the strong performance in the experiment is no surprise.

2 TECHNICAL THEOREMS AND HOW THEY COMBINE

We now formally state our main technical results, in their full generality, and show how they combine to yield Theorems 1, 2, and 3. A fair warning should be given to the reader. The theorems to follow are intricate and arguably somewhat inaccessible at first read. Rather than trying to understand everything at once, we suggest that the reader use this section as a roadmap for the full version of the paper. We will, however, do our best to explain the contents of the results as well as disentangling the various assumptions in the theorems.

As noted in Section 1.6, the exposition is subdivided into three parts, each yielding theorems that we believe to be of independent interest. First, we provide an improved analysis of simple tabulation. We then show how permuting the output of a simple tabulation hash function yields a hash function having Chernoff bounds for arbitrary value functions. Finally, we show that concatenating the output of two independent hash functions preserves the property of having Chernoff bounds for arbitrary value functions.

It turns out that the proofs of our results become a little cleaner when we assume that value functions take values in $[-1, 1]$, so from here on we state our results in relation to such value functions. Theorems 1, 2, and 3 will still follow, as the value functions in these theorems can also be viewed as having range $[-1, 1]$.

2.1 Improved Analysis of Simple Tabulation

Our new and improved result on simple tabulation is as follows. The proof can be found in the full version of the paper [2].

Theorem 4. *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function and $S \subseteq \Sigma^c$ be a set of keys of size $n = |S|$. Let $v\colon \Sigma^c \times [m] \to [-1, 1]$ be a value function such that the set $Q = \{i \in [m] \mid \exists x \in \Sigma^c : v(x, i) \neq 0\}$ satisfies $|Q| \leq m^\varepsilon$, where $\varepsilon < \frac{1}{4}$ is a constant.*

(1) *For any constant $\gamma \geq 1$, the random variable $V = \sum_{x \in S} v(x, h(x))$ is strongly concentrated with added error probability n/m^γ, where the constants of the asymptotics are determined by c and γ. Furthermore, this concentration is query invariant.*

(2) *For $j \in [m]$ define the random variable $V_j = \sum_{x \in S} v(x, h(x) \oplus j)$ and let $\mu = \mathbb{E}\left[V_j\right]$, noting that this is independent of j. For any $\gamma \geq 1$,*

$$\Pr\left[\sum_{j \in [m]} (V_j - \mu)^2 > D_{\gamma,c} \sum_{x \in S} \sum_{k \in [m]} v(x, k)^2\right] = O_{\gamma,\varepsilon,c}(n/m^\gamma) \tag{5}$$

for some constant $D_{\gamma,c}$ and this bound is query invariant up to constant factors.

The technical assumption involving Q states that the value function has *bounded support* in the hash range: The value $v(x, h(x))$ can only possibly be non-zero if $h(x)$ lies in the relatively small set Q of size at most m^ε. In fact, when proving Theorem 1 it suffices to assume that $|Q| = 1$, as we shall see below, but for our analysis of tabulation-permutation hashing we need the more general result above. Another nice illustration of the power of Theorem 4 holding with value functions of *any* bounded support will appear when we prove Theorem 3 in Section 2.4.

To see that Theorem 1 is implied by Theorem 4, one may observe that the latter is a generalization of the former. Let $y \in [m]$ be the bin and $(w_x)_{x\in S}$ be the weights of the balls from S in the setting of Theorem 1. Then defining the value function $v\colon \Sigma^c \times [m] \to [0, 1]$,

$$v(x, y') = \begin{cases} w_x \cdot [y' = y], & x \in S, \\ 0, & x \notin S, \end{cases}$$

we find that $X = \sum_{x\in S} w_x \cdot [h(x) = y] = \sum_{x\in S} v(x, h(x))$ is strongly concentrated by part 1 of Theorem 4 and the concentration is query invariant.

Finally, the bound (5) requires some explaining. For this, we consider the toy example of Theorem 1. Suppose we have a set $S \subseteq [u]$ of balls with weights $(w_x)_{x\in S}$ and we throw them into the bins of $[m]$ using a simple tabulation hash function. We focus on the total weight of balls landing in bin 0, defining the value function by $v(x, y) = w_x$ for $x \in S$ and $y = 0$, and $v(x, y) = 0$ otherwise. In this case, $\mu = \frac{1}{m}\sum_{x\in S} w_x$ denotes the expected total weight in any single bin and $V_j = \sum_{x\in S} w_x \cdot [h(x) = j]$ denotes the total weight in bin $j \in [m]$. Then (5) states that $\sum_{j\in[m]}(V_j - \mu)^2 = O(\|w\|_2^2)$ with high probability in m. This is exactly a bound on the *variance* of the weight of balls landing in one of the bins when each of the hash values of the keys of S are shifted by an XOR with a uniformly random element of $[m]$. Note that this example corresponds to the case where $|Q| = 1$. In its full generality, i.e., for general value functions of bounded support, (5) is similarly a bound on the variance of the value obtained from the keys of S when their hash values are each shifted by a uniformly random XOR. This variance bound turns out to be an important ingredient in our proof of the strong concentration in the first part of Theorem 4. As described in Section 1.6.1 the proof proceeds by fixing the hash values of the position characters $[c] \times \Sigma$ in a carefully chosen order, $\alpha_1 \prec \cdots \prec \alpha_r$. Defining G_i to be those keys that contain α_i as a position character but no α_j with $j > i$, the internal clustering of the keys of G_i is determined solely by $(h(\alpha_j))_{j<i}$ and fixing $h(\alpha_i)$ "shifts" each of these keys by an XOR with $h(\alpha_i)$. Now (5), applied with $S = G_i$, exactly yields a bound on the *variance* of the total value obtained from the keys from G_i when fixing the random XOR $h(\alpha_i)$. Thus, (5) conveniently bounds the variance of the martingale described in Section 1.6.1. As such, (5) is merely a technical tool, but we have a more important reason for including the bound in the theorem. As it turns out, for *any* hash function satisfying the conclusion of Theorem 4, composing with a uniformly random permutation yields a hash family having Chernoff-style concentration bounds for any value function as we describe next.

2.2 Permuting the Hash Range

Our next step in proving Theorem 2 is to show that, given a hash function with concentration bounds like in Theorem 4, composing with a uniformly random permutation of the entire range yields a hash function with Chernoff-style concentration for general value functions. The main theorem, proved in the full version of the paper [2], is as follows.

Theorem 5. *Let $\varepsilon \in (0, 1]$ and $m \geq 2$ be given. Let $g\colon [u] \to [m]$ be a 3-independent hash function satisfying the following. For every $\gamma > 0$, and for every value function $v\colon [u] \times [m] \to [-1, 1]$ such that the set $Q = \{i \in [m] \mid \exists x \in [u]\colon v(x, i) \neq 0\}$ is of size $|Q| \leq m^\varepsilon$, the two conclusions of Theorem 4 holds with respect to g.*

Let $v'\colon [u] \to [-1, 1]$ be any value function, $\tau\colon [m] \to [m]$ be a uniformly random permutation independent of g, and $\gamma > 0$. Then the for the hash function $h = \tau \circ g$, the sum $\sum_{x\in[u]} v'(x, h(x))$ is strongly concentrated, where the constants of the asymptotics are determined solely by c, d, and γ. Furthermore, this concentration is query invariant.

We believe the theorem to be of independent interest. From a hash function that only performs well for value functions supported on an asymptotically small subset of the bins we can construct a hash function performing well for any value function – simply by composing with a random permutation. Theorem 4 shows that simple tabulation satisfies the two conditions in the theorem above. It follows that if $m = |U|^{\Omega(1)}$, e.g., if $m = |\Sigma|$, then composing a simple tabulation hash function $g : \Sigma^c \to [m]$ with a uniformly random permutation $\tau : [m] \to [m]$ yields a hash function $h = \tau \circ g$ having Chernoff-style bounds for general value functions asymptotically matching those from the fully random setting up to an added error probability inversely polynomial in the size of the universe. In particular these bounds hold for tabulation-permutation hashing from Σ^c to Σ, that is, using just a single permutation, which yields the result of Theorem 2 in the case $d = 1$. If we desire a range of size $m \gg |\Sigma|$ the permutation τ becomes too expensive to store. Recall that in tabulation-permutation hashing from Σ^c to Σ^d we instead use d permutations $\tau_1, \ldots, \tau_d : \Sigma \to \Sigma$, hashing

$$\Sigma^c \xrightarrow{g^{\text{simple}}} \Sigma^d \xrightarrow{(\tau_1,\ldots,\tau_d)} \Sigma^d.$$

Towards proving that this is indeed sensible, the last step in the proof of Theorem 2 is to show that concatenating the outputs of independent hash functions preserves the property of having Chernoff-style concentration for general value functions.

2.3 Squaring the Hash Range

The third and final step towards proving Theorem 2 is showing that concatenating the hash values of two independent hash functions each with Chernoff-style bounds for general value functions yields a new hash function with similar Chernoff-style bounds up to constant factors. In particular it will follow that tabulation-permutation hashing has Chernoff-style bounds for general value functions. However, as with Theorem 5, the result is of more general interest. Since it uses the input hash functions in a black box manner, it is a general tool towards constructing new hash functions with Chernoff-style bounds. The main theorem, proved in the full version of the paper [2], is the following.

Theorem 6. *Let $h_1 : A \to B_1$ and $h_2 : A \to B_2$ be 2-wise independent hash functions with a common domain such that for every pair of value functions, $v_1 : A \times B_1 \to [-1,1]$ and $v_2 : A \times B_2 \to [-1,1]$, the random variables $X_1 = \sum_{a\in A} v_1(a, h_1(a))$ and $X_2 = \sum_{a\in A} v_2(a, h_2(a))$ are strongly concentrated with added error probability f_1 and f_2, respectively, and the concentration is query invariant. Suppose further that h_1 and h_2 are independent. Then the hash function $h = (h_1, h_2) : A \to B_1 \times B_2$, which is the concatenation of h_1 and h_2, satisfies that for every value function $v : A \times (B_1 \times B_2) \to [-1,1]$, the random variable $X = \sum_{a\in A} v(a, h(a)) = \sum_{a\in A} v(a, h_1(a), h_2(a))$ is strongly concentrated with additive error $O(f_1 + f_2)$ and the concentration is query invariant.*

We argue that Theorem 6, combined with the previous results, leads to Theorem 2.

Proof of Theorem 2. We proceed by induction on d. For $d = 1$ the result follows from Theorem 4 and 5 as described in the previous subsection. Now suppose $d > 1$ and that the result holds for smaller values of d. Let $\gamma = O(1)$ be given. Let $d_1 = \lfloor d/2 \rfloor$ and $d_2 = \lceil d/2 \rceil$. A tabulation-permutation hash function $h : \Sigma^c \to \Sigma^d$ is the concatenation of two independent tabulation-permutation hash functions $h_1 : \Sigma^c \to \Sigma^{d_1}$ and $h_2 : \Sigma^c \to \Sigma^{d_2}$. Letting $A = \Sigma^c$, $B_1 = \Sigma^{d_1}$, $B_2 = \Sigma^{d_2}$, the induction hypothesis gives that the conditions of Theorem 6 are satisfied and the conclusion follows. Note that since $d = O(1)$, the induction is only applied a constant number of times. Hence, the constants hidden in the asymptotics of Definition 1 are still constant. □

2.4 Concentration in Arbitrary Intervals.

We will now show how we can use our main result, Theorem 2, together with our improved understanding of simple tabulation Theorem 4 to obtain Theorem 3 which shows that the extra efficient tabulation-1permutation hashing provides Chernoff-style concentration for the special case of weighted balls and intervals. This section also serves as an illustration of how our previous results play in tandem, and it illustrates the importance of Theorem 4 holding, not just for single bins, but for any value function of bounded support.

Proof of Theorem 3. Let $S \subseteq [u]$ be a set of keys, with each key $x \in S$ having a weight $w_x \in [0,1]$. Let $h = \tau \circ g : \Sigma^c \to \Sigma^d = [r]$ be a tabulation-1permutation hash function, with $g : \Sigma^c \to \Sigma^d$ a simple tabulation hash function and $\tau : \Sigma^d \to \Sigma^d$ a random permutation of the most significant character, $\tau(z_1, \ldots, z_d) = (\tau_1(z_1), z_2, \ldots, z_d)$ for a uniformly random permutation $\tau_1 : \Sigma \to \Sigma$. Let $y_1, y_2 \in \Sigma^d$ and X be defined as in Theorem 3, $X = \sum_{x\in S} w_x \cdot [y_1 \le h(x) < y_2]$. Set $\mu = \mathbb{E}[X]$, and $\sigma^2 = \mathrm{Var}[X]$. For simplicity we assume that $|I| \ge r/2$. Otherwise, we just apply the argument below with I replaced by $[r] \setminus I = [0, y_1) \cup [y_2, r)$, which we view as an interval in the cyclic ordering of $[r]$. We will partition $I = [y_1, y_2)$ into a constant number of intervals in such a way that our previous results yield Chernoff style concentration bound on the total weight of keys landing within each of these intervals. The desired result will follow.

To be precise, let $t > 0$ and $\gamma = O(1)$ be given. Let $P_1 = \{x \in \Sigma \mid \forall y \in \Sigma^{d-1} : (x,y) \in I\}$ and $I_1 = \{(x_1, \ldots, x_d) \in \Sigma^d \mid x_1 \in P_1\}$. Whether or not $h(x) \in I_1$ for a key $x \in \Sigma^c$ depends solely on the most significant character of $h(x)$. With $X_1 = \sum_{x\in S} w_x \cdot [h(x) \in I_1]$, $\mu_1 = \mathbb{E}[X_1]$, and $\sigma_1^2 = \mathrm{Var}[X_1]$, we can therefore apply Theorem 2 to obtain that for any $t' > 0$ and $\gamma' = O(1)$,

$$\begin{aligned}\Pr[|X_1 - \mu_1| \ge t'] &\le C\exp(-\Omega(\sigma_1^2 C(t'/\sigma_1^2))) + 1/u^{\gamma'} \\ &\le C\exp(-\Omega(\sigma^2 C(t'/\sigma^2))) + 1/u^{\gamma'}, \qquad (6)\end{aligned}$$

for some constant C. Here we used that $\sigma_1^2 \le \sigma^2$ as $|I_1| \le |I| \le |\Sigma^d|/2$. Next, let $d_1 = \lg|\Sigma|$ and $d_2, \ldots, d_\ell \in \mathbb{N}$ be such that for $2 \le i \le \ell$, it holds that $2^{d_i} \le (2^{d_1+d_2+\cdots+d_i})^{1/4}$, and further $2^{d_1+d_2+\cdots+d_\ell} = |\Sigma|^d$. We may assume that u and hence $|\Sigma|$ is larger than some constant as otherwise the bound in Theorem 3 is trivial. It is then easy to check that we may choose ℓ and the $(d_i)_{2\le i\le \ell}$ such that $\ell = O(\log d) = O(1)$. We will from now on consider elements of Σ^d as numbers written in binary or, equivalently, bit strings of length $d' := d_1 + \cdots + d_\ell$. For $i = 1, \ldots, \ell$ we define a map $\rho_i : \Sigma^d \to [2]^{d_1+\cdots+d_i}$ as follows. If $x = b_1 \ldots b_{d'} \in [2]^{d'}$, then $\rho_i(x)$ is the length $d_1 + \cdots + d_i$ bit string $b_1 \ldots b_{d_1+\cdots+d_i}$ Set $J_1 = I$. For $i = 2, \ldots, \ell$ we define $J_i \subseteq I$ and $I_i \subseteq I$ recursively as follows. First, we let $J_i = J_{i-1} \setminus I_{i-1}$. Second, we define I_i to consist of those elements of $x \in J_i$ such that if $y \in \Sigma^c$ has $\rho_i(y) = \rho_i(x)$, then $y \in J_i$. In other words, I_i consists of those elements of J_i that remain in J_i when the least significant $d_{i+1} + \cdots + d_\ell$ bits of x are changed in an arbitrary manner. It is readily checked that for $i = 1, \ldots, \ell$, I_i is a disjoint union of two (potentially empty) intervals $I_i = I_i^{(1)} \cup I_i^{(2)}$ such that for each $j \in \{1,2\}$ and $x, y \in I_i^{(j)}$, $\rho_i(x) = \rho_i(y)$. Moreover, the sets $(I_i)_{i=1}^\ell$ are pairwise disjoint and $I = \bigcup_{i=1}^\ell I_i$.

We already saw in (6) that we have Chernoff-style concentration for the total weight of balls landing in I_1. We now show that the same is true for $I_i^{(j)}$ for each $i = 2, \ldots, \ell$ and $j \in \{0,1\}$. So let such an i and j be fixed. Note that whether or not $h(x) \in I_i^{(j)}$, for a key $x \in \Sigma^c$, depends solely on the most significant $d_1 + \cdots + d_i$ bits of $h(x)$. Let $h' : \Sigma^c \to [2]^{d_1+\cdots+d_i}$ be defined by $h'(x) = \rho_i(h(x))$. Then h' is itself a simple tabulation hash function and $h'(x)$ is obtained by removing the $d_{i+1} + \cdots + d_\ell$ least significant bits of $h(x)$. Letting $I' = \rho_i(I_i^{(j)})$, it thus holds that $h(x) \in I_i^{(j)}$ if and only if $h'(x) \in I'$. Let now $X_i^{(j)} = \sum_{x\in S} w_x \cdot [h(x) \in I_i^{(j)}]$, $\mu_i^{(j)} = \mathbb{E}\left[X_i^{(j)}\right]$, and $\sigma_1^2 = \mathrm{Var}\left[X_i^{(j)}\right] \le \sigma^2$. As $|I'| \le 2^{d_i} \le (2^{d_1+\cdots+d_i})^{1/4}$, we can apply Theorem 4 to conclude that for $t' > 0$ and $\gamma' = O(1)$,

$$\begin{aligned}\Pr[|X_i^{(j)} - \mu_i^{(j)}| \ge t'] &\le C\exp(-\Omega(\sigma_1^2 C(t'/\sigma_1^2))) + 1/u^{\gamma'} \\ &\le C\exp(-\Omega(\sigma^2 C(t'/\sigma^2))) + 1/u^{\gamma'}. \qquad (7)\end{aligned}$$

Now applying (6) and (7) with $t' = t/(2\ell - 1)$ and $\gamma' = \gamma + \frac{\log(2\ell)}{\log u} = O(1)$, it follows that

$$\begin{aligned}\Pr[|X - \mu| \ge t] &\le \Pr[|X_1 - \mu_1| \ge t'] + \sum_{i=2}^{\ell}\sum_{j=1}^{2} \Pr[|X_i^{(j)} - \mu_i^{(j)}| \ge t'] \\ &\le 2C\ell\exp(-\Omega(\sigma^2 C(t'/\sigma^2))) + 2\ell/u^{\gamma'} \\ &= O(\exp(-\Omega(\sigma^2\, C(t/\sigma^2)))) + 1/u^{\gamma},\end{aligned}$$

as desired. □

ACKNOWLEDGMENTS

Research supported by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

REFERENCES

[1] Anders Aamand, Evangelos Kipouridis, Jakob B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. 2020. No Repetitions: Fast Streaming with Highly Concentrated Hashing. *ArXiv* abs/2004.01156.

[2] Anders Aamand, Jakob Bæk Tejs Knudsen, Mathias B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. 2019. Fast hashing with Strong Concentration Bounds. *ArXiv* abs/1905.00369 (2019).

[3] Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup. 1996. Static Dictionaries on AC^0 RAMs: Query Time $\Theta(\sqrt{\log n / \log\log n})$ is Necessary and Sufficient. In *37th Annual Symposium on Foundations of Computer Science (FOCS)*. 441–450. https://doi.org/10.1109/SFCS.1996.548503

[4] Austin Appleby. 2016. *MurmurHash3*.

[5] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: Simpler, Smaller, Fast as MD5. In *Applied Cryptography and Network Security*, Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 119–135.

[6] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. 2002. Counting Distinct Elements in a Data Stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 1–10.

[7] George Bennett. 1962. Probability Inequalities for the Sum of Independent Random Variables. *J. Amer. Statist. Assoc.* 57, 297 (1962), 33–45. https://doi.org/10.1080/01621459.1962.10482149

[8] Sergei Natanovich Bernstein. 1924. On a modification of Chebyshev's inequality and of the error formula of Laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math.* 1 (1924), 38–49.

[9] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*. 21–29.

[10] Larry Carter and Mark N. Wegman. 1979. Universal classes of hash functions. *J. Comput. System Sci.* 18, 2 (1979), 143–154. Announced at STOC'77.

[11] L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. 2011. Balls and Bins: Smaller Hash Families and Faster Evaluation. In *52nd Annual Symposium on Foundations of Computer Science (FOCS)*. 599–608.

[12] Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. 1984. Constant Depth Reducibility. *SIAM J. Comput.* 13, 2 (1984), 423–439. https://doi.org/10.1137/0213028

[13] Herman Chernoff. 1952. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. *Annals of Mathematical Statistics* 23, 4 (1952), 493–507.

[14] Tobias Christiani and Rasmus Pagh. 2014. Generating k-Independent Variables in Constant Time. In *55th Annual Symposium on Foundations of Computer Science (FOCS)*. 196–205.

[15] Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. 2015. From independence to expansion and back again. In *Proceedings of the 47rd ACM Symposium on Theory of Computing (STOC)*.

[16] Kai-Min Chung, Michael Mitzenmacher, and Salil Vadhan. 2013. Why simple hash functions work: Exploiting the entropy in a data stream. *Theory of Computing* 9, 1 (2013), 897–945.

[17] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. 2015. Hashing for Statistics over K-Partitions. In *56th Annual Symposium on Foundations of Computer Science (FOCS)*. 1292–1310. https://doi.org/10.1109/FOCS.2015.83

[18] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. 2017. Practical Hash Functions for Similarity Estimation and Dimensionality Reduction. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*. Curran Associates Inc., 6618–6628. http://dl.acm.org/citation.cfm?id=3295222.3295407

[19] Martin Dietzfelbinger. 1996. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (STACS)*. 569–580.

[20] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. 1992. Dynamic Hashing in Real Time. In *Informatik, Festschrift zum 60. Geburtstag von Günter Hotz*. 95–119. https://doi.org/10.1007/978-3-322-95233-2_7

[21] Martin Dietzfelbinger and Michael Rink. 2009. Applications of a Splitting Trick. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*. 354–365.

[22] Martin Dietzfelbinger and Christoph Weidling. 2007. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.* 380 (06 2007), 47–68. https://doi.org/10.1016/j.tcs.2007.02.054

[23] Martin Dietzfelbinger and Philipp Woelfel. 2003. Almost Random Graphs with Simple Hash Functions. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*. 629–638.

[24] A. I. Dumey. 1956. Indexing for rapid random access memory systems. *Computers and Automation* 5, 12 (1956), 6–9.

[25] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. 2005. Space Efficient Hash Tables with Worst Case Constant Access Time. *Theory of Computing Systems* 38, 2 (01 Feb 2005), 229–248. https://doi.org/10.1007/s00224-004-1195-x

[26] Parikshit Gopalan, Daniel M. Kane, and Raghu Meka. 2018. Pseudorandomness via the Discrete Fourier Transform. *SIAM J. Comput.* 47, 6 (2018), 2451–2487. https://doi.org/10.1137/16M1062132

[27] Torben Hagerup and Torsten Tholey. 2001. Efficient Minimal Perfect Hashing in Nearly Minimal Space. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science (STACS)*. 317–326.

[28] John L. Hennessy and David A. Patterson. 2012. *Computer Architecture - A Quantitative Approach, 5th Edition*. Morgan Kaufmann.

[29] Donald E. Knuth. 1963. Notes on open addressing. (1963). Unpublished memorandum. See `http://citeseer.ist.psu.edu/knuth63notes.html`.

[30] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. 2003. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd Internet Measurement Conference (IMC)*. 234–247. https://doi.org/10.1145/948205.948236

[31] Daniel Lemire and Owen Kaser. 2016. Faster 64-bit universal hashing using carry-less multiplications. *Journal of Cryptographic Engineering* 6, 3 (01 Sep 2016), 171–185. https://doi.org/10.1007/s13389-015-0110-5

[32] Yishay Mansour, Noam Nisan, and Prasoon Tiwari. 1993. The Computational Complexity of Universal Hashing. *Theor. Comput. Sci.* 107, 1 (1993), 121–133. https://doi.org/10.1016/0304-3975(93)90257-T

[33] Raghu Meka, Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. 2014. Fast Pseudorandomness for Independence and Load Balancing - (Extended Abstract). In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*. 859–870.

[34] Peter Bro Miltersen. 1996. Lower Bounds for Static Dictionaries on RAMs with Bit Operations But No Multiplication. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP)*. 442–453. https://doi.org/10.1007/3-540-61440-0_149

[35] Michael Mitzenmacher and Salil P. Vadhan. 2008. Why simple hash functions work: exploiting the entropy in a data stream. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 746–755.

[36] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.

[37] Anna Pagh and Rasmus Pagh. 2008. Uniform Hashing in Constant Time and Optimal Space. *SIAM J. Comput.* 38, 1 (2008), 85–96.

[38] Mihai Pătraşcu and Mikkel Thorup. 2012. The Power of Simple Tabulation-Based Hashing. *J. ACM* 59, 3 (2012), Article 14. Announced at STOC'11.

[39] Mihai Pătraşcu and Mikkel Thorup. 2016. On the k-Independence Required by Linear Probing and Minwise Independence. *ACM Trans. Algorithms* 12, 1 (2016), 8:1–8:27.

[40] Geoff Pike and Jyrki Alakuijala. 2011. Introducing cityhash. https://opensource.googleblog.com/2011/04/introducing-cityhash.html.

[41] Mihai Pătraşcu and Mikkel Thorup. 2013. Twisted Tabulation Hashing. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 209–228.

[42] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. 1995. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics* 8, 2 (1995), 223–250. Announced at SODA'93.

[43] Alan Siegel. 2004. On Universal Classes of Extremely Random Constant-Time Hash Functions. *SIAM J. Comput.* 33, 3 (2004), 505–543. Announced at FOCS'89.

[44] Mikkel Thorup. 2013. Simple Tabulation, Fast Expanders, Double Tabulation, and High Independence. In *54th Annual Symposium on Foundations of Computer Science (FOCS)*. 90–99.

[45] Mikkel Thorup. 2015. High Speed Hashing for Integers and Strings. *ArXiv* abs/1504.06804 (2015).

[46] Mikkel Thorup and Yin Zhang. 2012. Tabulation-Based 5-Independent Hashing with Applications to Linear Probing and Second Moment Estimation. *SIAM J. Comput.* 41, 2 (2012), 293–331. Announced at SODA'04 and ALENEX'10.

[47] Mark N. Wegman and Larry Carter. 1981. New Classes and Applications of Hash Functions. *J. Comput. System Sci.* 22, 3 (1981), 265–279. Announced at FOCS'79.

[48] Albert Lindsey Zobrist. 1970. *A New Hashing Method with Application for Game Playing*. Technical Report 88. Computer Sciences Department, University of Wisconsin, Madison, Wisconsin.