Hashing for statistics over \( k \)-partitions

Søren Dahlgaard\(^1\), Mathias Bæk Tejs Knudsen\(^1\), Eva Rotenberg, and Mikkel Thorup\(^1\)

University of Copenhagen,
\{soerend, knudsen, roden, mthorup\}@di.ku.dk

Abstract

In this paper we propose a hash function for \( k \)-partitioning a set into bins so that we get good concentration bounds when combining statistics from different bins.

To understand this point, suppose we have a fully random hash function applied to a set \( X \) of red and blue balls. We want to estimate the fraction \( f \) of red balls. The idea of MinHash is to sample the ball with the smallest hash value. This sample is uniformly random and is red with probability \( f \). The standard method is to repeat the experiment \( k \) times with independent hash functions to reduce variance.

Consider the alternative experiment using a single hash function, where we use some bits of the hash value to partition \( X \) into \( k \) bins, and then use the remaining bits as a local hash value. We pick the ball with the smallest hash value in each bin.

The big difference between the two schemes is that the second one runs \( \Omega(k) \) times faster. In the first experiment, each ball participated in \( k \) independent experiments, but in the second one with \( k \)-partitions, each ball picks its bin, and then only participates in the local experiment for that bin. Thus, essentially, we get \( k \) experiments for the price of one. However, no realistic hash function is known to give the desired concentration bounds because the contents of different bins may be too correlated even if the marginal distribution for a single bin is random.

Here, we present and analyze a hash function showing that it does yields statistics similar to that of a fully random hash function when \( k \)-partitioning a set into bins. In this process we also give more insight into simple tabulation and show new results regarding the power of choice and moment estimation.

\(^1\)Research partly supported by Thorup’s Advanced Grant from the Danish Council for Independent Research under the Sapere Aude research carrier programme.

\(^2\)Research partly supported by the FNU project AlgoDisc - Discrete Mathematics, Algorithms, and Data Structures.
1 Introduction

A useful assumption in the design of randomized algorithms and data structures is the free availability of fully random hash functions which can be computed in unit time. Removing this assumption is the subject of a large body of work by providing realistic constructions of families of hash functions with desirable mathematical guarantees.

In this paper we consider the case where a hash function is used to $k$-partition a set into bins. This technique was introduced by Flajolet and Martin \cite{12} under the name stochastic averaging to estimate the number of distinct elements in a data stream. The goal is to get good concentration bounds when combining statistics from each bin.

To understand this point, suppose we have a fully random hash function applied to a set $X$ of red and blue balls. We want to estimate the fraction $f$ of red balls. The idea of MinHash is to sample the ball with the smallest hash value. This sample is uniformly random and is red with probability $f$. If we repeat the experiment $k$ times with $k$ independent hash functions, we get a multiset $S$ of $k$ samples with replacement from $X$. The fraction of red balls in $S$ concentrates around $f$ and the error probability falls exponentially in $k$.

Consider the alternative experiment using a single hash function, where we use some bits of the hash value to partition $X$ into $k$ bins, and then use the remaining bits as a local hash value. We pick the ball with the smallest hash value in each bin. This is a sample $S$ from $X$ without replacement, and again, the fraction of red balls is concentrated around $f$ with exponential concentration bounds. We note that there are some differences. We do get the advantage that the samples are without replacement, which means better concentration. On the other hand, we may end up with fewer samples if some bins are empty.

The big difference between the two schemes is that the second one runs $\Omega(k)$ times faster. In the first experiment, each ball participated in $k$ independent experiments, but in the second one with $k$-partitions, each ball picks its bin, and then only participates in the local experiment for that bin. Thus, essentially, we get $k$ experiments for the price of one. Handling each ball, or key, in constant time is important in applications of high volume streams.

We note that Bachrach and Porat \cite{3} have suggested a more efficient way of maintaining $k$ Minhash values with $k$ different hash functions. They use $k$ different polynomial hash functions that are related, yet pairwise independent, so that he can systematically maintain the Minhash for all $k$ polynomials in $O(\log k)$ time per key. This method is interesting but specialized for minwise hashing with polynomials. Also, because the experiments are only pairwise independent, the concentration is only limited by Chebyshev's inequality and not by Chernoff bounds.

The $k$-partition approach is generic and allows us to process keys in constant time for several types of statistics. One of the most classic applications of hash functions is to count the number of distinct elements in a data stream.
(i.e. estimating $F_0$). This is a problem, which has been studied in a wealth of work (see eg. [12, 13, 4]). A standard method is to use hashing to sample an element or hash value and use this sample to estimate $F_0$. As mentioned, this was the original application of $k$-partition in the seminal paper by Flajolet and Martin [12]. The technique has since been employed in the HyperLogLog counters of Flajolet et al. [13], to estimate the neighbourhood function of a graph with all-distance sketches [4, 10], in minwise sketch creation for large-scale learning [14, 20], and the count sketch data structure of Charikar et al. [9].

We note that our running example with red and blue balls is mathematically equivalent to the classic application of minwise hashing to estimate the Jaccard similarity $|X \cap Y|/|X \cup Y|$ between two sets $X$ and $Y$. This method was originally introduced by Broder et al. [7, 6, 5] for the AltaVista search engine. The red balls correspond to the intersection and the blue balls correspond to the symmetric difference. Addressing this application with $k$-partitions was suggested by Li et al. [14, 20]. In this work, $k$-partition is used to create small sketches of very high-dimensional indicator vectors. The sketches are then converted into vectors, which are fed to a linear SVM for classifying massive data sets. In this application it is important, that the $k$ samples of different sketches are aligned – that is, the sample of bin 1 of one sketch should be compared to the sample of bin 1 for another sketch. It is thus not possible to use other sketch variants such as the otherwise superior bottom-$k$ sketches (see e.g. [22]).

The issue we now have is that no realistic hashing scheme is known to give good probabilistic guarantees when $k$-partition is used for the above kind of statistics. The problem is that the contents of different bins may be too correlated, and then we get no better concentration with a larger $k$. This phenomenon is illustrated in Figure 1. In this figure we see how multiplication based methods may give highly correlated contents of different bins, even if an independent hash function was employed in each bin locally. It is not clear how relevant the classic independence paradigm of Wegman and Carter [25] is for general $k$-partitions if the independence is smaller than $k$.

Here we present and analyze a hash function showing that it does yield statistics similar to that of a fully random hash function in settings like those discussed above. We provide a general framework for analyzing such settings, without having to go into the details of the particular estimator considered.

The hash function we propose, mixed tabulation, is an efficient mix of simple tabulation and double tabulation. To analyze it we show that this hash function behaves like a truly random hash function on fairly large sets with high probability, even when some of the output bits of the hash function are known. In this process we also give more insight into both simple and double tabulation, defined below.

**Simple tabulation** In the simple tabulation scheme dating back to Zobrist [26] we view a key $x \in [u] = \{0, \ldots, u - 1\}$ as a vector of $c > 1$ characters $x_0, \ldots, x_{c-1} \in \Sigma = [u^{1/c}]$. The hash values are bit strings of some length $r$. 

---

2
\[ X = \frac{(1+0+1+0+0+0+1+0+0+0+0+0+0)}{11} \]

Figure 1: Estimating the fraction of red balls using 12-partition based on any hash function of the form \( h(x) = ax + b \mod p \). Blue balls have keys consecutive keys 0, 1, 2, ..., while red balls have random keys, hence random hash values.

For each character position, we initialize a fully random table \( T_i \) of size \(|\Sigma|\) with values from \( \mathbb{R} \). The hash value of a key \( x \) is calculated as

\[ h(x) = T_0[x_0] \oplus \cdots \oplus T_c-1[x_{c-1}] \cdot \]

Pătraşcu and Thorup [17] analyzed simple tabulation assuming \( c = O(1) \) in the context of several common applications of hash function such as linear probing, cuckoo hashing and minwise independence. We note that simple tabulation fails to give good concentration for \( k \)-partition: Consider the sets \( R = [2^\ell] \) and \( B = [2] \setminus ([n/2] \setminus [m/2]) \) for some \( m \leq n \). In this case all the elements of \( R \cup B \) will hash in pairs with probability \( 1/k \) (when \( T_0[0] = T_0[1] \)). Nevertheless we shall use the following result from [17, Theorem 1]:

**Theorem 1** (Pătraşcu and Thorup [17]). If we hash \( n \) keys into \( m \leq n \) bins with simple tabulation, then, with high probability (whp.\(^3\)) every bin gets \( n/m + O(\sqrt{n/m \log^c n}) \) keys.

We note that with simple tabulation hashing, the different output bit positions are completely independent of each other. We may thus pick any \( \ell \) bit positions in the hash values, and use them to define the \( m = 2^\ell \) bins in Theorem 1.

We say that a set of keys \( x_1, \ldots, x_k \) is independent if for any choice of \( h \), the hash values of any key \( x_i \) is independent of the hash values of the keys \( (x_j)_{j \neq i} \). If this is not the case we say that the set of keys is dependent.

\(^3\)With probability \( 1 - n^{-\gamma} \) for any \( \gamma = O(1) \).
Double tabulation  In double tabulation [23], we compose two independent simple tabulation functions \( h_1 : \Sigma^c \to \Sigma^d \) and \( h_2 : \Sigma^d \to \mathcal{R} \) defining \( h : \Sigma^c \to \mathcal{R} \) as \( h(x) = h_2(h_1(x)) \).

**Theorem 2** (Thorup [23]). If \( d \geq 6c \), then with probability \( 1 - o(|\Sigma|^{2-d/(2c)}) \) over the choice of \( h_1 \), the double tabulation function \( h_2 \circ h_1 \) is \( k = |\Sigma|^{1/(5c)} \) independent.

Thus, if we are not unlucky with \( h_1 \), we get that every subset \( X \subseteq [u] \) of at most \( k \) keys get completely independent hash values. Here we will prove:

**Theorem 3.** Given an arbitrary set \( S \subseteq [u] \) of size \( |S|/(1 + \Omega(1)) \), with probability \( 1 - |\Sigma|^{-[d/2]} \) over the choice of \( h_1 \), the double tabulation function \( h_2 \circ h_1 \) is fully random over \( S \).

It is interesting to compare Theorem 3 and Theorem 2. Theorem 3 holds with \( d = 4 \) “derived” characters gets essentially the same error probability as Theorem 2 with \( d = 6c \).

Siegel [21] has proved that with space \( |\Sigma| \) we cannot in constant time hope to get independence higher than \( |\Sigma|^{-\Omega(1)} \), which is much less than the size of the given set in Theorem 3.

Theorem 3 provides an extremely simple \( O(n) \) space implementation of a constant time hash function that is likely uniform on any given set \( S \). This should be compared with the uniform hashing of Pagh and Pagh [16].

We note, that their construction uses high-independent hash functions as a subroutine, and the current simplest implementation for constant time high independence is that of Theorem 2. On the other hand, the representation in [16] uses only \( (1 + \varepsilon)n \lg |\mathcal{R}| + O(n) \) bits whereas we use \( O(n \lg n + \lg |\mathcal{R}|) \) bits.

Mixed tabulation  In Theorem 3 we may use \( d = 4 \) even if \( c \) is larger, but then \( h_1 \) might introduce duplicate keys. To avoid this problem we mix the schemes in mixed tabulation. Mathematically, we use two simple tabulation hash functions \( h_1 : [u] \to \Sigma^d \) and \( h_2 : \Sigma^{c+d} \to \mathcal{R} \), and define the hash function \( h(x) \leftarrow h_2(x \circ h_1(x)) \), where \( \circ \) denotes concatenation of characters. We call \( x \circ h_1(x) \) the derived key. Since the derived keys include the original keys, there are no duplicate keys. This means that Theorem 4 holds for the application of \( h_2 \) to the derived keys.

We note that mixed tabulation only requires \( c+d \) lookups if we instead store simple tabulation functions \( h_{1,2} : \Sigma^c \to \Sigma^d \times \mathcal{R} \) and \( h'_2 : \Sigma^d \to \mathcal{R} \), computing \( h(x) \) by \((v_1, v_2) = h_{1,2}(x); h(x) = v_1 \oplus h_2(v_2)\). This efficient implementation is similar to that of twisted tabulation [18], and is equivalent to the previous definition.

We are going to show the following, more general version of Theorem 3.

**Theorem 4.** Let \( h \) be a mixed tabulation hash function consisting of two random simple tabulation hash functions \( h_1 : [u] \to \Sigma^d \), and \( h_2 : \Sigma^{c+d} \to \mathcal{R} \).
Define \( h^*_x \) as \( h^*_x(x) \rightarrow x \circ h_1(x) \). Let \( X \subseteq [u] \) be any input set. For each \( x \in X \), associate a function \( f_x : \mathcal{R} \rightarrow \{0, 1\} \), and let \( p_x \) be the probability that \( f_x(z) = 1 \) for \( z \) uniformly distributed in \( \mathcal{R} \).

Let \( Y = \{ x \in X \mid f_x(h(x)) = 1 \} \) and assume \( \mathbb{E}[|Y|] \leq |\Sigma|/(1 + \varepsilon) \). Then the keys of \( h^*_x(Y) \subseteq \Sigma^{\ell + d} \) are independent with probability \( 1 - O(|\Sigma|^{-[d/2]}) \).

As an example, recall that the output bits of \( h_2 \) are completely independent. Hence if the sampling functions \( f_x \) only depend on some of the output bits, then the hash function restricted to \( Y \) on the remaining bits will be fully random with probability \( 1 - O(|\Sigma|^{-[d/2]}) \). An extreme case is when \( f_x \) is a constant for each \( x \). Then we get fully random hash values on \( Y \) with probability \( 1 - O(|\Sigma|^{-[d/2]}) \). In particular, we get Theorem 3 as a corollary.

**Good statistics in few steps**

We are now going to present a general framework for understanding how mixed tabulation can be almost as good as fully random hashing for \( k \)-partitioning when \( k \leq |\Sigma|/(2d \log |\Sigma|) \). This means that space for the mixed tabulation tables only needs to be a logarithmic times bigger than the number of bins.

We will consider the example of MinHash with a set \( X \) of red and blue balls as described earlier. Recall that the hash value \( h(x) \) has two parts: one telling which of the \( k \) bins \( x \) lands in (i.e. the first \( \lg k \) bits) and the rest providing the local hash value, which we consider to be in \([0, 1]\).

Note first, that if \( |X| \leq |\Sigma|/2 \), then, by Theorem 3 mixed tabulation hashes all of \( X \) fully randomly with probability \( 1 - O(|\Sigma|^{-[d/2]}) \), and we are done, so we may assume \( |X| > |\Sigma|/2 \). The set \( X \) is partitioned into a set \( R \) of red balls and a set \( B \) of blue balls. We will assume \( |B| \geq |R| \), as the other case can be handled similarly. For \( C = R, B \), we will define a threshold probability \( p_C \) such that \( p_C|C| \approx \frac{|\Sigma|}{4} \) (or \( p_C = 1 \) if \( |C| \leq |\Sigma|/4 \)). Let \( S_C \) be the set of such balls from \( C \) with local hash value below \( p_C \). We then have:

1. \(|S_C| \) is sharply concentrated around \( p_C|C| \), i.e. \( |S_C| \approx p_C|C| \cdot (1 \pm \varepsilon) \) with \( \varepsilon = \tilde{O}(1/\sqrt{|\Sigma|}) \) by Theorem 1
2. The hash function \( h \) is fully independent on the remaining bits when restricted to \( S_C \) with probability \( 1 - O(|\Sigma|^{-[d/2]}) \) by Theorem 4
3. Every bin contains an element from \( S_B \) whp by point 2.

We will assume that \( p_C = 2^{-\ell_C} \) is a power of two. The difference between mixed tabulation and full randomness is thus restricted to the error probability of points 2 and 3 as well as the concentration bounds in point 1. The idea is illustrated in Figure 2.

So far we have ignored the interaction we get when we assign hash values to \( R \) and \( B \) with mixed tabulation. The concentration bounds are unchanged for they hold with high probability, hence for any constant number of cases. We first assign the \( \ell_R \) output bits of \( h_2 \) corresponding to thresholding by \( p_R \) (see Figure 2). Then the keys of \( h^*_R(S_R) \) are independent with probability
Figure 2: Illustration of the analysis for minwise hashing with mixed tabulation.

\[ h(x) = \overbrace{001010000000000101001010101}^{l_n} \]

We carry out the calculations in Section 4. By doing this we get the following theorem for minwise hashing with \( k \)-partition:

**Theorem 5.** Let \( h \) be a mixed tabulation hash function. Let \( Y \subseteq X \subseteq [u] \) be input sets. Let \( k \leq |\Sigma|/(2d \log |\Sigma|) \) be an input parameter. Let \( Z \) denote the number of bins in which the smallest hash value belongs to a red key, and let \( k' \) denote the number of non-empty bins. Then

\[
\Pr[Z/k' \geq (1 + \delta)f] \leq P_* + O \left( |\Sigma|^{1-|d/2|} \right),
\]

where \( P_* \) is the probability of the above error for the same experiment with \( |R|(1 + \varepsilon) \) red balls and \( |B|(1 - \varepsilon) \) blue balls using a truly random hash function with \( \varepsilon = O(\log^c |\Sigma|/\sqrt{|\Sigma|}) \).

We can get an analogous estimate for \( \Pr[Z/k' \leq (1 - \delta)f] \).

In particular, \( Z/k' \in f \pm \sqrt{f/k}\cdot\text{polylog}(\Sigma) \) with probability \( 1 - O(|\Sigma|^{1-|d/2|}) \).

Note that if \( |R|, |B| < |\Sigma|/3 \) we can eliminate the dependence on \( \varepsilon \).

We note that the sketches created in [14] are problematic in the case where there are empty bins, but in this case we get full randomness, which is the best we can hope for. In [20], they handle this problem by “borrowing” the sample value from the closest bin.
Other schemes  The same line of argument can be applied to the original counters of Flajolet and Martin [12] and the HyperLogLog counters of [13]. These cases are only simpler because we have just one color of balls. Particularly, with Hyperloglog, we are only interested in the smallest ball in each bin, and then it is analyzed like the blue balls above, but with no interference from red balls.

A different type of application is the count sketch of Charikar et al. [9], which provides a $k$-partition version of the second moment estimation of Alon et al. [1]. Using a 4-independent hash function, they get a relative standard deviation of $O(1/\sqrt{k})$, and then they take the median over $t$ independent experiments to reduce the error probability. With our method, unless we have really dominant items, we can get good concentration by increasing $k$ and avoid the parallel experiments, thus again getting down to constant time per key. This time we apply Theorem 4 to the $|\Sigma|/2$ largest values, concluding that they get fully random hash values. The remaining values are comparatively small, and can be handled with concentration bounds similar to those proved in [18] for collisions between a set of query keys and a set of stored keys.

1.1 Other results

In order to prove Theorem 4 we develop a number of structural lemmas in Section 2 relating to key dependencies in simple tabulation. These lemmas provides a basis for showing several interesting results for simple tabulation, which we also include in this paper. These results are briefly described below.

Constant moments  An alternative to Chernoff bounds in providing good concentration is to use bounded moments. We show that the $k$‘th moment of simple tabulation comes within a constant factor of that achieved by truly random hash functions for any constant $k$.

**Theorem 6.** Let $h : [u] \to \mathcal{R}$ be a simple tabulation hash function. Let $x_0, \ldots, x_{m-1}$ be $m$ distinct keys from $[u]$ and let $Y_0, \ldots, Y_{m-1}$ be any random variables such that $Y_i \in [0, 1]$ is a function of $h(x_i)$ with mean $E[Y_i] = p$ for all $i \in [m]$. Define $Y = \sum_{i \in [m]} Y_i$ and $\mu = E[Y] = mp$. Then for any constant integer $k \geq 1$:

$$E[(Y - \mu)^{2k}] = O\left(\sum_{j=1}^{k} \mu^j\right),$$

where the constant in the $O$-notation is dependent on $k$ and $c$.

The power of two choices  The power of two choices is a standard scheme for placing balls into bins, and is thoroughly studied in the literature (see [15] for a survey). When placing $m \leq n$ balls into $n$ bin$^2$ using the two-choice paradigm with truly random hash functions, the maximum load of any bin

$^2$We use a non standard notation, where $m$ denotes the number of balls and $n$ the number of bins to stay consistent with graph theory, where a graph has $m$ edges and $n$ vertices.
is $\lg \lg n + O(1)$ whp \[2\]. It was recently shown by Reingold et al. \[19\] how to guarantee a $O(\log \log n)$ bound on the maximum load whp using the hash functions of \[8\], which can be evaluated in $O((\log \log n)^2)$ time. It was claimed without proof in \[17\] that simple tabulation gives $O(\log \log n)$ expected load.

In Section 6 we prove significantly stronger bounds. We show that simple tabulation gives $\lg \lg n$ expected load, and $O(\log \log n)$ maximum load whp.

The first result relies on bounding the probability that a specific subgraph of a big binomial tree exists. Bounding the probability that a binomial tree exists is a standard technique, but to our knowledge restricting the argument to this subtree is new. The second result uses a generalized version of the lemmas of Section 2 to bound the probability of a large sub graph or a sub graph with high arboricity.

**Theorem 7.** Let $h_0$ and $h_1$ be two independent random simple tabulation hash functions. If $m = O(n)$ balls are put into two tables of $n$ bins sequentially using the two-choice paradigm with $h_0$ and $h_1$, then the expected maximum load is at most $\lg \lg n + O(1)$.

**Theorem 8.** Let $h_0$ and $h_1$ be two independent random simple tabulation hash functions. If $m = O(n)$ balls are put into two tables of $n$ bins sequentially using the two-choice paradigm with $h_0$ and $h_1$, then for any constant $\gamma > 0$ the maximum load of any bin is $O(\log \log n)$ with probability $1 - n^{-\gamma}$.

### 1.2 Notation

Let $S \subseteq [u]$ be a set of keys. Denote by $\pi(S, i)$ the projection of $S$ on the $i$th character, i.e. $\pi(S, i) = \{x_i | x \in S\}$. We also use this notation for keys, so $\pi((x_0, \ldots, x_{c-1}), i) = x_i$. A *position character* is an element of $[c] \times \Sigma$. Under this definition a key $x \in [u]$ can be viewed as a set of $c$ position characters $\{(0, x_0), \ldots, (c-1, x_{c-1})\}$. Furthermore we assume that $h$ is defined on position characters as $h((i, \alpha)) = T_i[\alpha]$. This definitions extends to sets of position characters in a natural way by taking the XOR over the hash of each position character. We denote the symmetric difference of the position characters of a set of keys $x_1, \ldots, x_k$ by

$$\bigoplus_{i=1}^k x_k.$$ 

The *hash graph* of hash functions $h_1 : [u] \rightarrow \mathcal{R}_1, \ldots, h_k : [u] \rightarrow \mathcal{R}_k$ and a set $S \subseteq [u]$ is the graph in which each element of $\mathcal{R}_1 \cup \ldots \cup \mathcal{R}_k$ is a node, and the nodes are connected by the (hyper-)edges $(h_1(x), \ldots, h_k(x)), x \in S$. In the graph there is a one-to-one correspondence between keys and edges, so we will not distinguish between those.
2 Bounding dependencies

The argument of Theorem 5 motivates the notion of uniform hashing for a set, which depends on the output values – i.e. the set of all keys hashing below some threshold $p$. This is captured in Theorem 4. In order to prove this theorem we need some structural lemmas regarding the dependencies of simple tabulation.

Simple tabulation is not 4-independent which means that there exists keys $x_1, ..., x_4$, such that for any choice of the random hash function $h$, $h(x_1)$ is dependent of $h(x_2), h(x_3), h(x_4)$. It was shown in [17], that for every sub-set $X \subseteq U$ with $|X| = n$ there are at most $O(n^2)$ such dependent 4-tuples $(x_1, x_2, x_3, x_4) \in X^4$.

In this section we show that a similar result holds in the case of dependent $k$-tuples, which is one of the key ingredients in the proofs of the main theorems of this paper.

We know from [24] that if the keys $x_1, ..., x_k$ are dependent, then there exists a non-empty subset $I \subseteq \{1, ..., k\}$ such that

$$\bigoplus_{i \in I} x_i = \emptyset.$$  

Following this observation we wish to bound the number of tuples which have symmetric difference $\emptyset$.

**Lemma 1.** Let $X \subseteq U$ with $|X| = n$ be a subset. The number of $2t$-tuples $(x_1, ..., x_{2t}) \in X^{2t}$ such that

$$x_1 \oplus \cdots \oplus x_{2t} = \emptyset$$

is at most $(2t-1)!!^c n^t$, where $(2t-1)!! = (2t-1)(2t-3) \cdots 1$.

It turns out that it is more convenient to prove the following more general lemma.

**Lemma 2.** Let $A_1, \ldots, A_{2t} \subseteq U$ be sets of keys. The number of $2t$-tuples $(x_1, ..., x_{2t}) \in A_1 \times \cdots \times A_{2t}$ such that

$$x_1 \oplus \cdots \oplus x_{2t} = \emptyset$$  

is at most $(2t-1)!!^c \prod_{i=1}^{2t} \sqrt{|A_i|}$.

**Sketch of proof.** The main idea in the proof is to bound the number of ways we can chose the keys $(x_1, ..., x_{2t})$ by bounding the number of ways we can chose the position characters. Since there must be an even number of each position character we can match them in pairs and $2t$ elements can be paired up in at most $(2t-1)!!$ ways. By using Cauchy-Schwartz and induction on $c$ we get the result. The idea is illustrated in Figure 3.

**Proof of Lemma 2.** Let $(x_1, ..., x_{2t})$ be such a $2t$-tuple. Equation (1) implies that the number of times each position character appears is an even number.
Figure 3: Pairing of the position characters of $2t$ keys. $x_1^{(0)}$ can be matched to $2t-1$ position characters, $x_2^{(0)}$ to $2t-3$, etc.

Hence we can partition $(x_1, \ldots, x_{2t})$ into $t$ pairs $(x_{i_1}, x_{j_1}), \ldots, (x_{i_t}, x_{j_t})$ such that $\pi(x_{i_k}, c-1) = \pi(x_{j_k}, c-1)$ for $k = 1, \ldots, t$. Note that there are at $(2t-1)!!$ ways to partition the elements in such a way.

We now prove the claim by induction on $c$. First assume that $c = 1$. We fix some partition $(x_{i_1}, x_{j_1}), \ldots, (x_{i_t}, x_{j_t})$ and count the number of $2t$-tuples which satisfy $\pi(x_{i_k}, c-1) = \pi(x_{j_k}, c-1)$ for $k = 1, \ldots, t$. Since $c = 1$ we have $x_{i_k}, x_{j_k} \in A_{i_k} \cap A_{j_k}$. The number of ways to choose such a $2t$-tuple is thus bounded by:

$$\prod_{k=1}^{t} |A_{i_k} \cap A_{j_k}| \leq \prod_{k=1}^{t} \min \{|A_{i_k}|, |A_{j_k}|\} \leq \prod_{k=1}^{t} \sqrt{|A_{i_k}| |A_{j_k}|} = \prod_{k=1}^{2t} \sqrt{|A_k|}$$

And since there are $(2t-1)!!$ such partitions the case $c = 1$ is finished.

Now assume that the lemma holds when the keys have $< c$ characters. As before, we fix some partition $(x_{i_1}, x_{j_1}), \ldots, (x_{i_t}, x_{j_t})$ and count the number of $2t$-tuples which satisfy $\pi(x_{i_k}, c-1) = \pi(x_{j_k}, c-1)$ for all $k = 1, \ldots, t$. Fix the last position character $(a_k, c-1) = \pi(x_{i_k}, c-1) = \pi(x_{j_k}, c-1)$ for $k = 1, \ldots, t$, $a_k \in \Sigma$. The rest of the position characters from $x_{i_k}$ is then from the set

$$A_{i_k}[a_k] = \{x \backslash (a_k, c-1) \mid (a_k, c-1) \in x, x \in A_{i_k}\}$$

By the induction hypothesis the number of ways to choose $x_1, \ldots, x_{2t}$ with this choice of $a_1, \ldots, a_t$ is then at most:

$$((2t-1)!!)^{c-1} \prod_{k=1}^{t} \sqrt{|A_{i_k}[a_k]| |A_{j_k}[a_k]|}$$
Summing over all choices of $a_1, \ldots, a_t$ this is bounded by:

$$
((2t - 1)!!)^{c-1} \sum_{a_1, \ldots, a_t \in \Sigma} \prod_{k=1}^t \sqrt{|A_{i_k} [a_k]| |A_{j_k} [a_k]|}
$$

$$
= ((2t - 1)!!)^{c-1} \prod_{k=1}^t \sum_{a_k \in \Sigma} \sqrt{|A_{i_k} [a_k]| |A_{j_k} [a_k]|}
$$

$$
\leq ((2t - 1)!!)^{c-1} \prod_{k=1}^t \sqrt{\sum_{a_k \in \Sigma} |A_{i_k} [a_k]| \sum_{a_k \in \Sigma} |A_{j_k} [a_k]|}
$$

$$
= ((2t - 1)!!)^{c-1} \prod_{k=1}^t \sqrt{|A_{i_k}| |A_{j_k}|} = ((2t - 1)!!)^{c-1} \prod_{k=1}^{2t} \sqrt{|A_k|}
$$

Here (2) is an application of Cauchy-Schwartz’s inequality. Since there are $(2t - 1)!!$ such partitions the conclusion follows.

Note that Lemma 1 does not bound the number of dependent 2t-tuples, but only the ones which are minimal in the sense that any proper subset of the 2t-tuple is not dependent. We can use Lemma 1 to bound the number of dependent tuples.

**Lemma 3.** Let $X \subseteq U$ with $|X| = n$ be a subset and fix $s$ such that $s^c \leq \frac{4}{5} n$. The number of s-tuples $(x_1, \ldots, x_s) \in X^s$ for which there exists $y \in X \setminus \{x_1, \ldots, x_s\}$ such that $h(y)$ is dependent on $h(x_1), \ldots, h(x_s)$ is at most

$$
s^{4 \frac{3c}{6}} n^{s-1} = s^{O(1)} n^{s-1}.
$$

**Proof.** Since $h(y)$ is dependent of $h(x_1), \ldots, h(x_s)$ there exists a subset $I \subseteq \{1, \ldots, s\}$ such that for all choices of $h$:

$$
\bigoplus_{i \in I} x_i = y
$$

Fix $|I|$ and note that $|I| \geq 3$ (by 3-independence). There are $\binom{s}{|I|}$ ways to choose $I$. Note that $(x_i)_{i \in \{1, \ldots, s\} \setminus I}$ can be chosen in at most $n^s - |I|$ ways, and by Lemma 1 $(x_i)_{i \in I}$ can be chosen in at most $((|I|)!!)^c n^{(|I|+1)/2}$ ways. I.e. for a fixed value of $|I|$, an upper bound is:

$$
\binom{s}{|I|} ((|I|)!!)^c n^{s-|I|/2 + 1/2}
$$

We can show that this upper bound is maximal when $|I| = 3$. Since $|I|$ is odd it suffices to show that the value decreases when $|I|$ increases by 2 as long as $|I| + 2 \leq s$. Consider the following fraction:

$$
\frac{\binom{s}{|I|} ((|I|)!!)^c n^{s-|I|/2 + 1/2}}{\binom{s}{|I|+2} ((|I|+2)!!)^c n^{s-(|I|+2)/2 + 1/2}} = \frac{(|I| + 1)(|I| + 2)n}{(s - |I|)(s - |I| - 1)(|I| + 2)^c} \geq \frac{\frac{3}{2} n}{s^c}
$$
By the assumption this fraction is at least 1, and hence the upper bound decreases with $|I|$. Therefore, as $|I|$ grows, there are fewer ways to describe $(x_1, \ldots, x_s)$.

Since $3 \leq |I| \leq s$, the number of ways to choose $(x_1, \ldots, x_s)$ is bounded from above by:

$$s \cdot \binom{s}{3} (3!!)^c n^{s-3/2 + 1/2} \leq s \cdot \frac{3^c}{6} n^{s-1}$$

□

The result of Lemma 3 only deals with a single key $y$. The following lemma shows that it is even more unlikely to have many such keys $y$.

**Lemma 4.** Let $X \subseteq U$ with $|X| = n$ and fix $s$ such that $s^c \leq \frac{4}{5} n$. The number of $s$-tuples $(x_1, \ldots, x_s)$ for which there exists distinct $y_1, \ldots, y_k \in X \setminus \{x_1, \ldots, x_s\}$ for $k \geq \max(s - 1, 5)$ such that each $h(y_i)$ is dependent on $h(x_1), \ldots, h(x_s)$ is at most

$$s^6 \frac{15c}{120} n^{s-2} + s^6 \frac{9c}{36} n^{s-2} + s^5 \frac{9c}{4} n^{s-3/2} = s^6 \frac{15c}{120} n^{s-2} \cdot O(1) \cdot n^{s-3/2}.$$

**Proof.** For each $j = 1, \ldots, k$ let $I_j \subseteq \{1, \ldots, s\}$ be such that $y_j = \bigoplus_{i \in I_j} x_i$ for all choices of $h$. All the tuples for which $|I_j| > 3$ for some $j$ can be bounded easily using the same idea as in Lemma 3. The upper bound decreases as $|I_j|$ increases, and since $|I_j| \geq 5$ we can use (3) to get an upper bound on these $s$-tuples which is

$$s^6 \frac{15c}{120} n^{s-2} \cdot O(1) \cdot n^{s-3/2}.$$

Now assume that $|I_j| = 3$ for $j = 1, \ldots, k$. Note that the sets $I_j$ must be distinct and since $I_j \subseteq \{1, \ldots, s\}$ and $k \geq \max\{5, s - 1\}$ there must exist $j, l \in \{1, \ldots, k\}$ such that $|I_j \cap I_l| \leq 1$.

Case $|I_j \cap I_l| = 0$: In this case the number of possible values for $(x_i)_{i \in I_j}$, $(x_i)_{i \in I_l}$, $I_j$, and $I_l$ is, by Lemma 3, no more than:

$$\binom{s}{3, 3} (3!!)^c n^{2} \cdot \left(3!! \cdot n^{2}\right)^2$$

and the remaining $x_i$’s can be chosen in at most $n^{s-6}$ ways giving an upper bound of:

$$s^6 \otimes (3!!)^c n^{2} \cdot \left(3!! \cdot n^{2}\right)^2 \cdot n^{s-6} = s^6 \frac{9c}{36} n^{s-2} \cdot (3!!)^c n^{2}.$$

Case $|I_j \cap I_l| = 1$: $I_j$ and $I_l$ can be chosen in $s \cdot 3$ ways. By Lemma 4 $(x_i)_{i \in I_j}$ can be chosen in $(3!!)^c n^{2}$ ways. The number of ways to choose $(x_i)_{i \in I_l}$ once $(x_i)_{i \in I_j}$ is then by Lemma 3 no more than $(3!!)^c n^{3/2}$ since we choose one of the $A_i$’s to be a singleton. The remaining $x_i$’s can be chosen in at most $n^{s-5}$ ways giving a total upper bound of:

$$s^6 \otimes (3!!)^c n^{2} \cdot (3!!)^c n^{3/2} \cdot n^{s-5} \leq s^5 \frac{9c}{4} n^{s-3/2}.$$
Which concludes the proof. □

3 Uniform hashing in constant time

This section is dedicated to prove Theorem 4. Here, we consider only the case when there exists a $p$ such that $p_x = p$ for all $x$. We note that the full proof uses the same ideas, but is more technical. See Section 3.1 for the details.

The proof is structured in the following way: (1) We fix $Y$ and assume the key set $h_i^*(Y)$ is not independent. (2) With $Y$ fixed this way we construct a bad event. (3) We unfix $Y$ and show that the probability of a bad event occurring is low using a union bound. Each bad event consists of independent “sub-events” relating to subgraphs of the hash graph of $h_1(Y)$. These sub-events fall into four categories, and for each of those we will bound the probability that the event occurs.

First observe that if a set of keys $S$ consists of independent keys, then the set of keys $h_i^*(S)$ are also independent.

We will now describe what we mean by a bad event. We consider the hash function $h_i : [u] → \Sigma^d$ as $d$ simple tabulation hash functions $h^{(0)}, \ldots, h^{(d-1)} : [u] → \Sigma$ and define $G_{i,j}$ to be the hash graph of $h^{(i)}, h^{(j)}$ and the input set $X$.

Fix $Y$ and consider some $y \in Y$. If for some $i,j$, the component of $G_{i,j}$ containing $y$ is a tree, then we can perform a peeling process and observe that $h_i^*(y)$ must be independent of $h_i^*(Y \setminus \{y\})$. Now assume that there exists some $y_0 \in Y$ such that $h_i^*(y_0)$ is dependent of $h_i^*(Y \setminus \{y_0\})$, then $y_0$ must lie on a (possibly empty) path leading to a cycle in each of $G_{i,i+1}$ for $i \in [\lfloor d/2 \rfloor]$. We will call such a path and cycle a lollipop. Denote this lollipop by $y_0, y_1, y_2, \ldots, y_i$. For each such $i$ we will construct a list $L_i$ to be part of our bad event. Set $s \overset{def}{=} \lceil 2\log_{1+\epsilon}|\Sigma| \rceil$. The list $L_i$ is constructed in the following manner: We walk along $y_1, \ldots, y_i$ until we meet an obstruction. Consider a key $y_j$. We will say that $y_j$ is an obstruction if it falls into one of the following four cases as illustrated in Figure 4.

A There exists some subset $B \subseteq \{y_0, y_1, \ldots, y_{i-1}\}$ such that $y_j = \bigoplus_{y \in B} y$.

B If case A does not hold and there exists some subset $B \subseteq \{y_0, y_1, \ldots, y_{j-1}\} \cup L_0 \cup \ldots \cup L_{i-1}$ such that $y_j = \bigoplus_{y \in B} y$.

C $j = p_i < s$ (i.e. $y_j$ is the last key on the cycle). In this case $y_j$ must share a node with either $y_0$ (the path is empty) or with two of the other keys in the lollipop.

D $j = s$. In this case the keys $y_1, \ldots, y_s$ form a path keys independent from $L_0, \ldots, L_{i-1}$.

In all four cases we set $L_i = (y_1, \ldots, y_s)$ and we associate an attribute $A_i$. In case A we set $A_i = B$. In case B we set $A = (x^{(0)}, \ldots, x^{(s-1)})$, where $x^{(r)} \in B$ is chosen such that $t_{(y_j, r)} = t_{(x^{(r)}, r)}$. In C we set $A_i = z$, where $z$ is the
smallest value such that $y'_j$ shares a node with $y'_i$, and in $D$ we set $A_i = \emptyset$. Denote the lists by $L$ and the types and attributes of the lists by $T, A$. We have shown, that if there is a dependency among the keys of $h_i(Y)$, then we can find such a bad event $(y_0, L, T, A)$.

Figure 4: The four types of violations. Dependent keys are denoted by ■ and •.

Now fix $y_0 \in X, l = (l_0, \ldots, l_{[d/2]-1})$. Let $F(y_0, l)$ be the event that there exists a quadruple $(y_0, L, T, A)$ forming a bad event such that $|L_i| = l_i$. We use the shorthand $F = F(y_0, l)$. Let $F(y_0, L, T, A)$ denote the event that a given quadruple $(y_0, L, T, A)$ occurs. Note that a quadruple $(y_0, L, T, A)$ only occurs if some conditions are satisfied for $h_1$ (i.e. that the hash graph forms the lollipops as described earlier) and $h_2$ (i.e. that the keys of the lollipops are contained in $Y$). Let $F_1(y_0, L, T, A)$ and $F_2(y_0, L, T, A)$ denote the the event that those conditions are satisfied, respectively. Then

$$
\Pr[F] \leq \sum_{\text{bad event } L, T, A} \Pr[F(y_0, L, T, A)] = \sum_{\text{bad event } L, T, A} \Pr[F_2(y_0, L, T, A)|F_1(y_0, L, T, A)] \cdot \Pr[F_1(y_0, L, T, A)] .
$$

We note, that $F_1(y_0, L, T, A)$ consists of independent events for each $G_{2i, 2i+1}$ for $i \in [[d/2]]$. Denote these restricted events by $F_1^i(y_0, L, T, A)$.

For a fixed $h_i$ we can bound $\Pr[F_2(y_0, L, T, A)]$ in the following way: For each $i \in [[d/2]]$ we choose a subset $V_i \subseteq L_i$ such that $S = \{y_0\} \cup V_i$ consists of independent keys. Since these keys are independent, so is $h_i^*(S)$, so we can bound the probability that $S \subseteq Y$ by $p^{|S|}$. We can split this one part for each $i$. Define

$$
p_i \overset{\text{def}}{=} p^{|V_i|} \cdot \Pr[F_1^i(y_0, L_i, T_i, A_i)] .
$$

We can then bound $\Pr[F] \leq p \cdot \prod_{i \in [[d/2]]} p_i$.
We now wish to bound the probability \( p_i \). Consider some \( i \in [[d/2]] \). We split the analysis into a case for each of the four types:

**A** Let \( \Delta(y_0) \) be the number of triples \((a, b, c) \in X^3\) such that \( y_0 \oplus a \oplus b \oplus c = \emptyset \). Note that the size of the attribute \(|A_i| \geq 3\) must be odd. Consider the following three cases:

1. \(|A_i| = 3, y_0 \in A_i\): We have \( y_0 \) is the \( \oplus \)-sum of three elements of \( L_i \). The number of ways this can happen (i.e. the number of ways to choose \( L_i \) and \( A_i \)) is bounded by \( l_i^3 n_i^{l_i-3} \Delta(y_0) \) – The indices of the three summands can be chosen in at most \( l_i^3 \) ways, and the corresponding keys in at most \( \Delta(y_0) \) ways. The remaining elements can be chosen in at most \( n_i^{3-l_i} \) ways.

2. \(|A_i| \geq 5, y_0 \in A_i\): By Lemma 2 we can choose \( L_i \) and \( A_i \) in at most \( l_i^{O(1)} \cdot n_i^{l_i-5/2} \) ways.

3. \(|A_i| \geq 3, y_0 \notin A_i\): By Lemma 2 we can choose \( L_i \) and \( A_i \) in at most \( l_i^{O(1)} \cdot n_i^{l_i-2} \) ways.

To conclude, we can choose \( L_i \) and \( A_i \) in at most

\[
l_i^{O(1)} \cdot n_i^{l_i-2} \cdot \left(1 + \frac{\Delta(y_0)}{n}\right)
\]

ways. We can choose \( V_i \) to be \( L_i \) except for the last key. We note that \( V_i \cup \{y_0\} \) form a path in \( G_{2i,2i+1} \), which happens with probability \( 1/|\Sigma|^{l_i} \) since the keys are independent. For type A we thus get the bound

\[
p_i \leq l_i^{O(1)} \cdot p^{l_i-1} \cdot n_i^{l_i-2} \cdot \left(1 + \frac{\Delta(y_0)}{n}\right) \cdot \frac{1}{|\Sigma|^{l_i-1}}
\]

\[
\leq l_i^{O(1)} \cdot \left(1 + \frac{\Delta(y_0)}{n}\right) \cdot \frac{1}{|\Sigma|} \cdot \frac{p}{(1 + \varepsilon)^{l_i-2}}
\]

\[
\leq l_i^{O(1)} \cdot \left(1 + \frac{\Delta(y_0)}{n}\right) \cdot \frac{1}{|\Sigma|} \cdot \frac{1}{(1 + \varepsilon)^{l_i/2}}.
\]

**B** All but the last key of \( L_i \) are independent and can be chosen in at most \( n_i^{l_i-1} \) ways. The last key is uniquely defined by \( A_i \), which can be chosen in at most \( p^{l_i} \) ways (where \( l = \sum_i l_i \)), thus \( L_i \) and \( A_i \) can be chosen in at most \( n_i^{l_i-1} p^{l_i} \) ways. Define \( V_i \) to be all but the last key of \( L_i \). The keys of \( L_i \cup \{y_0\} \) form a path, and since the last key of \( L_i \) contains a position character not in \( V_i \), the probability of this path occurring is exactly \( 1/|\Sigma|^{l_i} \), thus we get

\[
p_i \leq p^{l_i} \cdot n_i^{l_i-1} \cdot \frac{1}{|\Sigma|^{l_i}} \leq p^{O(1)} \cdot \frac{1}{|\Sigma|} \cdot \frac{1}{(1 + \varepsilon)^{l_i-1}} \leq p^{O(1)} \cdot \frac{1}{|\Sigma|} \cdot \frac{1}{(1 + \varepsilon)^{l_i/2}}.
\]
C The attribute $A_i$ is just a number in $[l_i]$, and $L_i$ can be chosen in at most $n^{l_i}$ ways. We can choose $V_i = L_i$. $V_i \cup \{y_0\}$ is a set of independent keys forming a path leading to a cycle, which happens with probability $1/|\Sigma|^{l_i+1}$, so we get the bound

$$p_i \leq l_i \cdot n^{l_i} \cdot p^{l_i} \cdot \frac{1}{|\Sigma|^{l_i+1}} \leq l_i \cdot \frac{1}{|\Sigma|} \cdot \frac{1}{(1+\varepsilon)^{l_i}} \leq p^{O(1)} \cdot \frac{1}{|\Sigma|} \cdot \frac{1}{(1+\varepsilon)^{l_i/2}}.$$

D The attribute $A_i = \emptyset$ is uniquely chosen. $L_i$ consists of $s$ independent keys and can be chosen in at most $n^s$ ways. We set $V_i = L_i$. We get

$$p_i \leq n^s \cdot p^s \cdot \frac{1}{|\Sigma|^s} \leq \frac{1}{(1+\varepsilon)^s} \cdot \frac{1}{|\Sigma|} \cdot \frac{1}{(1+\varepsilon)^{l_i/2}}.$$

We first note, that there exists $y_0$ such that $\Delta(y_0) = O(n)$. We have just shown that for a specific $y_0$ and partition of the lengths $(l_0, \ldots, l_{\lfloor d/2 \rfloor})$ we get

$$\Pr[F] \leq p \cdot \left( p^{O(1)} \cdot \frac{1}{|\Sigma|} \right)^{\lfloor d/2 \rfloor} \cdot \frac{1}{(1+\varepsilon)^{l_i/2}}.$$

Summing over all partitions of the $l_i$'s and choices of $l$ gives

$$\sum_{l \geq 1} p \cdot |\Sigma|^{-\lfloor d/2 \rfloor} \cdot \frac{1}{(1+\varepsilon)^{l_i/2}} \leq O \left( p \cdot |\Sigma|^{-\lfloor d/2 \rfloor} \right).$$

We have now bounded the probability for $y_0 \in X$ that $y_0 \in Y$ and $y_0$ is dependent on $Y \setminus \{y_0\}$. We relied on $\Delta(y_0) = O(n)$, so we cannot simply take a union bound. Instead we note that, if $y_0$ is independent of $Y \setminus \{y_0\}$ we can peel $y_0$ away and use the same argument on $X \setminus \{y_0\}$. This gives a total upper bound of

$$O \left( \sum_{y_0 \in X} p \cdot |\Sigma|^{-\lfloor d/2 \rfloor} \right) = O(\Sigma^{-\lfloor d/2 \rfloor}).$$

This finishes the proof. $\square$

3.1 Uniform hashing with multiple probabilities

Here we present a sketch of the proof of an extension to Theorem 4. We only need to handle the four different cases where we bound $p_i$. First, we argue that cases B, C, and D are handled in almost the exact same way. In the original proof we argued that for some size $v$ we can choose $V_i$, $|V_i| = v$ in at most $n^v$ ways and for each choice of $V_i$ the probability that it is contained in $Y$ is at most $p^v$, thus multiplying the upper bound by

$$n^v p^v = (E|Y|)^v.$$

For our proof we sum over all choices of $V_i$ and add the probabilities that $V_i$ is contained in $Y$ getting the exact same estimate:

$$\sum_{V_i \in U, |V_i| = v} \left( \prod_{x \in V_i} p_x \right)^v \leq \left( \sum_{x \in U} p_x \right)^v = (E|Y|)^v.$$
Next, and this is the difficult part, we prove the claim in case A.

For all \( i \geq 0 \) we set
\[
n_i = \left| \left\{ x \in X \mid p_x \in \{2^{-i-1}, 2^{-i}\} \right\} \right|.
\]
Now observe, that \( \sum_{i \geq 0} n_i 2^{-i} \leq 2 |\Sigma| / (1 + \varepsilon) = O(|\Sigma|) \). Define \( m_i = \sum_{j \leq i} n_j \); we then have:
\[
\sum_{i \geq 0} m_i 2^{-i} = \sum_{i \geq 0} n_i \left( \sum_{j \geq 1} 2^{-j} \right) = \sum_{i \geq 0} n_i 2^{-i+1} = O(|\Sigma|)
\]
We let \( X_i = \{ x \in X \mid p_x > 2^{-i-1} \} \) and note that \( m_i = |X_i| \). For each \( y_0 \in X \) we will define \( \Delta'(y_0) \) (analogously to \( \Delta(y_0) \)) in the following way:
\[
\Delta'(y_0) = \min_{a,b,c \in X} \{ p_a p_b p_c p_d p_e \}
\]
where we only sum over triples \( (a, b, c) \) such that \( y_0 \oplus a \oplus b \oplus c = \emptyset \). Analogously to the original proof we will show that there exists \( y_0 \) such that \( \Delta'(y_0) \leq O(|\Sigma|) \). The key here is to prove that:
\[
\sum_{y_0 \in X} \Delta'(y_0) = O(n |\Sigma|)
\]
Now consider a 4-tuple \( (y_0, a, b, c) \) such that \( y_0 \oplus a \oplus b \oplus c = \emptyset \). Let \( i \geq 0 \) be the smallest non-negative integer such that \( b, c \in X_i \). Then:
\[
\min_{a,b,c \in X} \{ p_a p_b p_c p_d p_e \} \leq \min \{ p_b, p_c \} \leq 2^{-i}
\]
By Lemma 2, we see that for any \( i \) there are at most \( O(n m_i) \) 4-tuples \( (y_0, a, b, c) \) such that \( b, c \in X_i \). This gives the following bound on the total sum:
\[
\sum_{y_0 \in X} \Delta'(y_0) \leq \sum_{i \geq 0} O(n m_i) \cdot 2^{-i} = O(n |\Sigma|)
\]
Hence there exists \( y_0 \) such that \( \Delta'(y_0) = O(|\Sigma|) \) and we can finish case A.1 analogously to the original proof.

Now we turn to case A.2 where \( |A_i| \geq 5 \), \( y_0 \in A_0 \). Here, we will only consider the case \( |A_i| = 5 \), since the other cases follow by the same reasoning. We will choose \( V_i \) to consist of all of \( L_i \setminus A_i \) and 3 keys from \( A_i \). We will write \( A_i = \{a, b, c, d, e\} \) and find the smallest \( \alpha, \beta, \gamma \) such that \( a, b \in X_\alpha, c, d \in X_\beta, e \in X_\gamma \). Then:
\[
\prod_{x \in V_i} p_x \leq \left( \prod_{x \in V_i \setminus A_i} p_x \right) 2^{-\alpha} 2^{-\beta} 2^{-\gamma}
\]
When $a,b \in X_\alpha, c,d \in X_\beta, e \in X_\gamma$ we can choose $a,b,c,d,e$ in at most $m_\alpha m_\beta \sqrt{m_\gamma}$ ways by Lemma \ref{lem:2}. Hence, when we sum over all choices of $V_i$ we get an upper bound of:

$$\left( \sum_{x \in X} p_x \right)^{l_5} \left( \sum_{a, \beta, \gamma \geq 0} m_\alpha m_\beta \sqrt{m_\gamma} 2^{-\alpha} 2^{-\beta} 2^{-\gamma} \right)$$

$$= \left( \sum_{x \in X} p_x \right)^{l_5} \left( \sum_{a \geq 0} m_\alpha 2^{-a} \right)^2 \left( \sum_{a \geq 0} \sqrt{m_\alpha} 2^{-a} \right)$$

Now we note that by Cauchy-Schwartz inequality:

$$\sum_{a \geq 0} \sqrt{m_\alpha} 2^{-a} \leq \sqrt{\sum_{a \geq 0} 2^{-a}} \sqrt{\sum_{a \geq 0} m_\alpha 2^{-a}} = O(\sqrt{|\Sigma|})$$

Hence we get a total upper bound of $O(|\Sigma|^{l_5/2})$ and we can finish the proof in analogously to the original proof.

Case A.3 is handled similarly to A.2.

### 4 Minwise hashing with $k$-partition

Here we carry out the calculations preceding Theorem \ref{thm:5}. We will assume that $|R| \geq |\Sigma|/4, |B| \geq |\Sigma|/2$ in order to simplify the calculations. We choose $p_R$ and $p_B$ such that:

$$\mathbb{E}[|S_B|] = p_B |B| \in \left( \frac{|\Sigma|}{4}, \frac{|\Sigma|}{2} \right), \mathbb{E}[|S_R|] = p_R |R| \in \left( \frac{|\Sigma|}{8}, \frac{|\Sigma|}{4} \right)$$

We state three properties about $S_C, C \in \{R, B\}$:

1. $|S_C|$ is sharply concentrated around $p_C|C|$, i.e. $|S_C| \in p_C|C| \cdot (1 \pm \epsilon)$ with $
\epsilon = \tilde{O}(1/\sqrt{|\Sigma|})$ by Theorem \ref{thm:1}

2. The hash function $h$ is fully independent on the remaining bits when restricted to $S_C$ with probability $1 - O(|\Sigma|^{1-|d/2|})$ by Theorem \ref{thm:4}

3. Every bin contains an element from $S_B$ whp by point 2.

We choose $\epsilon$ large enough such that the probability that $|S_C| \in p_C|S_C| \cdot (1 \pm \epsilon)$ is $1 - O(|\Sigma|^{1-|d/2|})$. In order to argue that all the the three points are satisfied with probability $1 - O(|\Sigma|^{1-|d/2|})$ we only need to show that if point 1 and 2 are satisfied then point 3 is satisfied with probability $1 - O(|\Sigma|^{1-|d/2|})$. In this case $S_B$ contains at least $\frac{|\Sigma|}{4}(1 - \epsilon)$ elements, and since the bits deciding which bin each ball goes to are fully random, the probability that there exists a bin
for some constant $c$.

$p | Z$ can now calculate the probability on the choice of $B$ such that probability that the remaining bits of the hash values are decided fully randomly. We let $h_1(S_R)$.

$E_3$: There is a dependency among the keys hashing below the threshold $p_B$. $E_5$: There exists a bin which contains no key from $S_B$.

The probability that any of the bad events happen is $O\left(|\Sigma|^{1-|d|/2}\right)$, and we will give an upper bound $P$ assuming that none of the bad events happen - this bound will at most be $O\left(|\Sigma|^{1-|d|/2}\right)$ to small.

First we will fix the $\ell_B$ bits of $h_2$ which decide $S_R$. We can then fix the $\ell_B - \ell_R$ bits of $h_2$ which will decide $S_B$. Note that $S_B$ might depend on $S_R$, but since $E_3$ does not happen we know that $|S_B| \geq (1 - \epsilon)p_B |B|$. Conditioned on the choice of $S_B$ and the number of red balls below the threshold $p_B$, we can now calculate the probability $\frac{|Z|}{K} \geq (1 + \delta)f$ by using the fact that $E_4$ and $E_5$ do not happen. Since the probability that $\frac{|Z|}{K} \geq (1 + \delta)f$ increases as $|S_B|$ decreases we can upper bound this probability assuming that there are $(1 - \epsilon)p_B |B|$ blue balls below $p_B$.

Hence we now consider the experiment where we have $S_R$ red balls below $p_R$ and exactly $(1 - \epsilon)p_B |B|$ blue independent balls below $p_B$. Now we note that probability that $\frac{|Z|}{K} \geq (1 + \delta)f$ increases as $|S_R|$ increases, and hence we consider the following experiment: There are exactly $(1 + \epsilon)p_R |R|$ red balls hashing below $p_R$ and exactly $(1 - \epsilon)p_B |B|$ blue balls hashing below $p_B$ such that the remaining bits of the hash values are decided fully randomly. We let $P'$ denote the probability that there $\frac{|Z|}{K} \geq (1 + \delta)f$. We have proved that

$$P \leq P' + O\left(|\Sigma|^{1-|d|/2}\right)$$

It remains to prove that $P' \leq P_* + O\left(|\Sigma|^{1-|d|/2}\right)$. We let $\epsilon_1 = c_0 \cdot \sqrt{\frac{\log |\Sigma|}{|K|}}$ for some constant $c_0 > 0$ to be decided. We then consider the experiment where we have $(1 + \epsilon)(1 + \epsilon_1) |R|$ red balls and $(1 - \epsilon)(1 - \epsilon_1) |B|$ blue balls,
such that the balls are placed using a fully random hash-function and let $P_*$ denote the probability $\frac{Z}{k} \geq (1 + \delta)\hat{f}$ for this experiment. This correspond to the $P_*$ in the statement of the theorem since $(1 + \varepsilon)(1 + \varepsilon) = (1 + \varepsilon')$ for some $\varepsilon' = O\left(\frac{\log|\Sigma|}{\sqrt{|\Sigma|}}\right)$ and similarly for $(1 - \varepsilon)(1 - \varepsilon)$.

By the same reasoning we used earlier we see that with probability $1 - O\left(|\Sigma|^{-1/d/2}\right)$ there exist a ball hashing below $p_B$ in each bin. Now we choose $c_0$ big enough then with probability $1 - O\left(|\Sigma|^{-1/d/2}\right)$ there are $\leq (1 - \varepsilon)p_B |B|$ blue balls hashing below $p_B$ and $\geq (1 + \varepsilon)p_R |R|$ red balls hashing below $p_R$. In this case the probability that $\frac{Z}{k} \geq (1 + \delta)\hat{f}$ is larger than $P'$, hence:

$$P_* \geq P' - O\left(|\Sigma|^{-1/d/2}\right)$$

This concludes the proof.

5 Constant moment bounds

This section is dedicated to proving Theorem 6.

Consider first Theorem 6 and let $k = O(1)$ be fixed. Define $Z_i = Y_i - p$ for all $i \in [m]$ and $Z = \sum_{i \in [m]} Z_i$. We wish to bound $E[Z^{2k}]$ and by linearity of expectation this equals:

$$E[Z^{2k}] = \sum_{r_0, \ldots, r_{2k-1} \in [m]^{2k}} E[Z_{r_0} \cdots Z_{r_{2k-1}}]$$

Fix some $2k$-tuple $r = (r_0, \ldots, r_{2k-1}) \in [m]^{2k}$ and define $V(r) = E[Z_{r_0} \cdots Z_{r_{2k-1}}]$. Observe, that if there exists $i \in [2k]$ such that $x_{r_i}$ is independent of $(x_{r_j})_{j \neq i}$ then

$$V(r) = E[Z_{r_0} \cdots Z_{r_{2k-1}}] = E[Z_{r_i}] E[\prod_{j \neq i} Z_{r_j}] = 0$$

The following lemma bounds the number of $2k$-tuples $r$ with $V(r) \neq 0$.

**Lemma 5.** The number of $2k$-tuples $r$ such that $V(r) \neq 0$ is $O(m^k)$.

**Proof.** Fix $r \in [m]^{2k}$ and let $T_0, \ldots, T_{s-1}$ be all subsets of $[2k]$ such that $\bigoplus_{i \in T_j} x_{r_i} = \emptyset$ for $j \in [s]$. If $\bigcup_{j \in [s]} T_j \neq [2k]$ we must have $V(r) = 0$, as there exists some $x_{r_i}$ that is independent of $(x_{r_j})_{j \neq i}$. Thus, we can assume that $\bigcup_{j \in [s]} T_j = [2k]$.

Now, fix $T_0, \ldots, T_{s-1} \subseteq [2k]$ such that $\bigcup_{j \in [s]} T_j = [2k]$, and count the number of ways to choose $r \in [m]^{2k}$ such that $\bigoplus_{i \in T_j} x_{r_i} = \emptyset$ for all $j \in [s]$. Note that $T_0, \ldots, T_{s-1}$ can be chosen in at most $2^{2k} = O(1)$ ways, so if we can bound the number of ways to choose $r$ by $O(m^k)$ we are done. Let $A_i = \bigcup_{j < i} T_j$ and $B_i = T_i \setminus A_i$ for $i \in [s]$. We will choose $r$ by choosing $(x_{r_i})_{i \in B_0}$,
then \((x_{r_i})_{i \in B_1}\), and so on up to \((x_{r_i})_{i \in B_{j-1}}\). When we choose \((x_{r_i})_{i \in B_j}\), we have already chosen \((x_{r_i})_{i \in A_j}\) and by Lemma \([2]\) the number of ways to choose \((x_{r_i})_{i \in B_j}\) is bounded by:

\[
(|T_j| - 1)! |m|^{B_j}/2 = O\left(|m|^{B_j}/2\right)
\]

Since \(\bigcup_{j \in [q]} B_j = [2k]\), we conclude that the number of ways to choose \(r\) such that \(V(r) \neq 0\) is at most \(O(m^k)\). \(\square\)

We note that since \(|V(r)| \leq 1\) this already proves that

\[
E[Z^{2k}] \leq O(m^k)
\]

Consider now any \(r \in [m]^{2k}\) and let \(f(r)\) denote the size of the largest subset \(I \subseteq [2k]\) of independent keys \((x_{r_i})_{i \in I}\). We then have

\[
E \left[ \prod_{i \in [2k]} Z_{r_i} \right] \leq E \left[ \prod_{i \in [2k]} Z_{r_i} \right] \leq E \left[ \prod_{i \in I} Z_{r_i} \right] \leq O\left( p^{f(r)} \right)
\]

We now fix some value \(s \in \{1, \ldots, 2k\}\) and count the number of \(2k\)-tuples \(r\) such that \(f(r) = s\). We can bound this number by first choosing the \(s\) independent keys of \(I\) in at most \(m^s\) ways. For each remaining key we can write it as a sum of a subset of \((x_{r_i})_{i \in I}\). There are at most \(2^s = O(1)\) such subsets, so there are at most \(O(m^s)\) such \(2k\)-tuples \(r\) with \(f(r) = s\).

Now consider the \(O(m^k)\) \(2k\)-tuples \(r \in [m]^{2k}\) such that \(V(r) \neq 0\). For each \(s \in \{1, \ldots, 2k\}\) there are \(O(m^{\min\{k,s\}})\) ways to choose \(r\) such that \(f(r) = s\). All these choices of \(r\) satisfy \(V(r) \leq O(p^s)\). Hence:

\[
E[Z^{2k}] = \sum_{r \in [m]^{2k}} V(r) \leq 2k \sum_{s=1}^{2k} O(m^{\min\{k,s\}}) \cdot O(p^s) = O\left( \sum_{s=1}^{k} (pm)^s \right).
\]

This finishes the proof of Theorem 9. \(\square\)

A similar argument can be used to show the following theorem, where the bin depends on a query key \(q\).

**Theorem 9.** Let \(h: [u] \rightarrow \mathcal{R}\) be a simple tabulation hash function. Let \(x_0, \ldots, x_{m-1}\) be \(m\) distinct keys from \([u]\) and let \(q \in [u]\) be a query key distinct from \(x_0, \ldots, x_{m-1}\). Let \(Y_0, \ldots, Y_{m-1}\) be any random variables such that \(Y_i \in [0,1]\) is a function of \((h(x_i), h(q))\) and for all \(r \in \mathcal{R}\), \(E[Y_i | h(q) = r] = p\) for all \(i \in [m]\). Define \(Y = \sum_{i \in [m]} Y_i\) and \(\mu = E[Y] = mp\). Then for any constant integer \(k \geq 1:\n
\[
E[(Y - \mu)^{2k}] \leq O\left( \sum_{j=1}^{k} \mu^j \right),
\]

where the constant in the \(O\)-notation is dependent on \(k\) and \(c\).
6 Power of two choices

This section is dedicated to proving Theorems 7 and 8.

6.1 Maximum load with high probability

This section is dedicated to proving Theorem 8. The main idea of the proof is to show that a hash graph resulting in high load must either have a huge component, or a component with high arboricity. We show that both cases are very unlikely.

As a negative result, we will first observe, that we cannot prove that the maximum load is \( \lg \lg n + O(1) \) or even \((1 + o(1)) \lg \lg n \) whp. when using simple tabulation.

**Observation 1.** Given \( k = O(1) \), there exists an ordered set \( S \) consisting of \( n \) keys, such that when they are distributed into \( n \) bins using hash values from simple tabulation the max load is \( \geq \left[ k^{c-1}/2 \right] \lg \lg n - O(1) \) with probability \( \Omega(n^{-2(k-1)(c-1)}) \).

**Proof.** Consider the set of keys \([n/k^{c-1}] \times [k]^{c-1}\) consisting of \( n \) keys. For each of the positions \( i = 1, \ldots, c-1 \) the probability that all the position characters on position \( i \) hash to the same value is \( n^{-k+1} \). So with probability \( n^{-(k-1)(c-1)} \) this happens for all positions \( i = 1, \ldots, c-1 \). This happens for both hash functions with probability \( n^{-2(k-1)(c-1)} \). In this case \( h_l(x) = h_l(x_0x_1 \ldots x_{c-1}) + h_l(x_0 \ldots x_{c-2}) \) is only dependent on \( h_l(x_{c-1}), l \in \{0, 1\} \). Order the keys lexicographically and insert them into the bins. If \( n/k^{c-1} = \Omega(n) \) balls are distributed independently and uniformly at random to \( n \) bins the maximum load would be \( \lg \lg n - O(1) \) with probability \( \Omega(1) \).

(This can be proved along the lines of [2, Thm. 3.2].) If we had exactly \( 2 \left[ k^{c-1}/2 \right] \) copies of \( n/k^{c-1} \) independent and random keys the maximum load would be at least \( \left[ k^{c-1}/2 \right] \) times larger than if we had had \( n/k^{c-1} \) independent and random keys. The latter is at least \( \lg \lg n - O(1) \) with probability \( \Omega(1) \).

Since there are \( k^{c-1} \geq 2 \left[ k^{c-1}/2 \right] \) copies of independent and uniformly random hash values we conclude that the maximum load is at least \( \left[ k^{c-1}/2 \right] \lg \lg n - O(1) \) with probability \( \Omega(1) \) under the assumption that \( h_l(x_0 \ldots x_{c-2}) \) is constant for any \( (x_0, \ldots, x_{c-2}) \in [k]^{c-1}, l \in \{0, 1\} \). Since the latter happens with probability \( n^{-2(k-1)(c-1)} \) the proof is finished.

We will now show that a series of insertions inducing a hash graph with low arboricity and small components cannot cause a too big maximum load. Note that this is the case for any hash functions and not just simple tabulation.

**Lemma 6.** Consider the process of placing some balls into bins with the two choice paradigm, and assume that some bin gets load \( k \). Then there exists a
connected component in the corresponding hash graph with \( x \) nodes and arboricity \( a \) such that:

\[
a \lg x \geq k
\]

Proof. Let \( v \) be the node in the hash graph corresponding to the bin with load \( k \). Let \( V_k = \{v\}, E_k = \emptyset \) and define \( V_l, E_l \) for \( l, 0 \leq l < k \) in the following way: For each bin of \( V_{l+1} \), add the edge corresponding to the \( l + 1 \)th ball landing in the bin to the set \( E_l \). Define \( V_l \) to be the endpoints of the edges in \( E_l \) (see Figure 5 for a visualization).

![Figure 5: A visualisation of the sets \( V_0, \ldots, V_k \).](image)

It is clear, that each bin of \( V_l \) must have a load of at least \( l \). Note that the definition implies that \(|E_l| = |V_{l+1}| \) and \( V_k \subseteq V_{k-1} \subseteq \ldots \subseteq V_0 \). For each \( l \in [k] \) consider the subgraph \((V_l, E_l \cup E_{l+1} \cup \ldots \cup E_{k-1})\) and let \( a_l \) be defined as the following lower bound on the arboricity of this subgraph:

\[
a_l = \left\lceil \frac{|E_l| + \ldots + |E_{k-1}|}{|V_l| - 1} \right\rceil
\]

Let \( a = \max_{l \in [k]} a_l \), then \( a \) is a lower bound on the arboricity of \((V_0, E_0 \cup \ldots \cup E_{k-1})\). Now note that for each \( l \in [k] \):

\[
\frac{|E_l| + \ldots + |E_{k-1}|}{|V_l| - 1} \leq a
\]

Since \(|E_l| = |V_{l+1}| \) for each \( l \in [k] \) this means that:

\[
|V_l| - 1 \geq \frac{|V_{l+1}| + \ldots + |V_k|}{a}
\]

By an easy induction \(|V_l| \geq \left(1 + \frac{1}{a}\right)^{k-l} \), and therefore \(|V_0| \geq \left(1 + \frac{1}{a}\right)^k \). The connected component that contains \( v \) contains at least \(|V_0|\) nodes, has arboricity \( \geq a \), and:

\[
a \lg |V_0| \geq a \lg \left(1 + \frac{1}{a}\right)^k = k \lg \left(1 + \frac{1}{a}\right)^a \geq k
\]

\(\square\)
In order to show that components cannot be too large or have too big arboricity, we will need to generalize some of the lemmas from Section 2. We will need the following combinatorial lemma.

**Lemma 7.** Let $s, k, c \geq 1$ be integers and $A_1, \ldots, A_{(2k)^c s + 1}$ be non-empty subsets of $\{1, \ldots, s\}$, such that for every $B \subseteq \{1, \ldots, s\}$:

$$|\{A_i \mid A_i \subseteq B\}| \leq |B|^c$$

Then there exists $I \subseteq \{1, \ldots, (2k)^c s + 1\}$ such that $|I| \leq k$ and

$$f(I) \stackrel{\text{def}}{=} \left| \bigcup_{i \in I} A_i \right| - |I| \geq k$$

**Proof.** Let $I \subseteq \{1, \ldots, (2k)^c s + 1\}$ be such that $|\cup_{i \in I} A_i| < 2k$. We want to show that there exists $J = I \cup \{r\}$ for some $r \in \{1, \ldots, (2k)^c s + 1\}$ such that $f(J) > f(I)$. Let $A = \cup_{i \in I} A_i$ and assume for the sake of contradiction that no such $r$ exists. This implies that $|A_i \setminus A| \leq 1$ for all $r \in \{1, \ldots, (2k)^c s + 1\}$. I.e. each $A_r$ is contained in one of the sets

$$(A \cup \{1\}), (A \cup \{2\}), \ldots, (A \cup \{s\})$$

By assumption, each of these sets contains no more than $(|A| + 1)^c$ sets $A_r$, and thus they contain at most $(|A| + 1)^c s$ sets combined. This means that

$$(2k)^c s + 1 \leq (|A| + 1)^c s \leq (2k)^c s,$$

which is a contradiction. Thus there must exists an $r$ such that $f(I \cup \{r\}) > f(I)$.

Now consider the following greedy algorithm: Let $I := \emptyset$ and iteratively set $I := I \cup \{r\}$ for such an $r$ until $|\cup_{i \in I} A_i| \geq 2k$. Since $f(I)$ increases in each step, the algorithm stops after at most $k$ steps. This implies that $f(I) \geq 2k - k = k$ and $|I| \leq k$ as desired. □

We can use Lemma 7 to show a more general version of Lemma 4.

**Lemma 8.** Let $X \subseteq U$ be a subset with $n$ elements and fix $k = O(1)$ and $s$ such that $ks^{kc} < \sqrt{n}$. The number of $s$-tuples $(x_1, \ldots, x_s) \in X^s$ for which there exists distinct $y_1, \ldots, y_{(2k)^c s + 1} \in X$, which are dependent on $x_1, \ldots, x_s$ is no more than:

$$n^{s-k/2} s^{O(1)}$$

where the constant in the $O$-notation is dependent on $k$.

**Proof.** For each $i \in \{1, \ldots, (2k)^c s + 1\}$ let $A_i \subseteq \{1, \ldots, s\}$ be such that:

$$\bigoplus_{j \in A_i} x_j = y_i$$
By Lemma [7], there exists $I \subseteq \{1, \ldots, (2k)^c + 1\}$ such that for $A := \cup_{i \in I} A_i$, $|A| - |I| \geq k$, $|I| \leq k$.

It is enough to show the lemma for a fixed $|A|$ and $|I|$ as these can be chosen in at most $ks = O(s)$. Fix $|A| = a$ and $|I| = r$.

Let $I = \{v_1, \ldots, v_r\}$ and for each $j \in \{1, \ldots, r\}$ define $B_j$ as:

$$B_j = A_{v_j} \setminus \left( \bigcup_{i < j} A_{v_i} \right)$$

Wlog. assume that $a = \sum_{j < r} |B_j| \leq 2k$. (Otherwise there exists a smaller set $I$) The number of ways to choose $(B_j)_{1 \leq j \leq r}$ is at most $s^r$: There are $(s)$ ways to choose $A$ and $r^a$ ways to partition $A$ into $B_1, \ldots, B_r$.

Now, fix the choice of $B_1, \ldots, B_r$. We will bound the number of ways to choose $(x_i)_{i \in B_j}$ given that $(x_i)_{i \in B_1, \ldots, (x_i)_{i \in B_{j-1}}}$ are chosen. The number of ways to choose $A_j$ is at most $2^{2k}$ for $j \in I$. For a fixed choice of $A_j$ the number of ways to choose $(x_i)_{i \in B_j}$ is at most $(|A_j|!!)^c n^{(|B_j|+1)/2}$ by Lemma [2]. Hence, the number of ways to choose $(x_i)_{i \in A}$ is at most:

$$\prod_{j=1}^r \left( 2^{2k} |A_j|!! \right)^c n^{(|B_j|+1)/2} \leq 2^{2kr} (a!!)^r n^{(a+r)/2} \leq 2^{2k} (a!!) n^{a+2+r/2}$$

The number of ways to choose the remaining $(x_i)_{i \in A}$ is trivially bounded by $n^{r-a}$ giving a total upper bound on the number of ways to choose $(x_i)_{i \in \{1, \ldots, s\}}$ of:

$$\binom{s}{a} k^a 2^{2k} (a!!) n^{s-a/2+r/2}$$

Now note that if $a < s$:

$$\binom{s}{a+1} 2^{2k} \binom{(a + 1)!}{k} n^{s-(a+1)/2+r/2} \leq \binom{s}{a} \frac{2^{2k} (a!!)}{k} n^{s-a/2+r/2} = \frac{(s - a) k (a + 1)^c}{(a + 1)n^{1/2}} < 1$$

This implies that the upper bound is biggest when $a$ is smallest, i.e. when $a = r + k$. In this case the upper bound is:

$$\left( \binom{s}{k} k^k 2^{2k} (r + k)!! \right) n^{s-k/2} \leq \left( \binom{s}{k} k^k 2^{2k} (2k)!! \right) n^{s-k/2} = s^O(1) n^{s-k/2}$$

which concludes the proof.

**Lemma 9.** Let $X \subseteq [u]$ with $|X| = m$, and let $h_0, h_1 : [u] \to [n]$ be two independent simple tabulation hash functions. Fix some integer $k$. If $m < n/(2^k(4k)^c)$, then the maximum load of any bin when assigning keys using the two-choice paradigm is $O(\log \log n)$ with probability $1 - O(n^{-k+2})$.

**Proof.** Fix the hash values of all the keys and consider the hash graph. Note that there is a one-to-one correspondence between the edges and the keys and we will not distinguish between the two in this proof. Consider any connected
subgraph $C$ in the hash graph. We wish to argue that $C$ cannot be too big or have too high arboricity. In order to do this, we construct a set $S$ of independent edges contained in $C$. Initially let $S = \{e\}$ for some edge in $e$ in $C$. At all times we maintain the set $Y = Y(S)$ of keys which are dependent on the keys in $S$. Note that $S \subseteq Y$. The set $S$ is constructed iteratively in the following way: If there exists an edge $e \in C \setminus Y$ that is incident to an edge in $S$ add $e$ to $S$. Otherwise, if there exists an edge $e \in C \setminus Y$, which is incident to an edge in $Y$, add $e$ to $S$. If neither type of edge exists we do not add more edges to $S$. Note that in this case $C = Y$.

At any point we can partition the edges of $S$ into connected components $C_1, \ldots, C_t$, such that $C_1$ is the component of the initial edge of $S$. For each $i > 1$ we let $b_i \in Y \setminus S$ be an edge incident to $C_i$. Order the components $C_2, \ldots, C_t$ such that $b_2 < \ldots < b_t$. For a visualisation of $S$ Figure 6 can be consulted.

![Figure 6: A visualization of the process. $C_1, \ldots, C_t$ correspond to components, and the red, dashed lines correspond to edges $b_2, \ldots, b_t$. For a visualisation of $S$ Figure 6 can be consulted.](image)

We stop the algorithm when either $|S| \geq k \lg n$ or $|Y| > (4k)^c |S|$. We will show that the probability that this can happen in the hash graph is bounded by $O(n^{-k+2})$. The two cases are described below and the proof of each case is ended with a $\alpha$.

**The algorithm stops because $|Y| > (4k)^c |S|$**: In this case we know that $|S| \leq k \lg n$ since the algorithm has not stopped earlier. Fix the size $|S| = s$ and the number of components $t$. First we bound the number of ways we can choose the subgraphs $C_1, \ldots, C_t$. Let $a_i$ be the number of nodes in the subgraph $C_i$. We can choose the structure of a spanning tree in each of $C_1, \ldots, C_t$ in no more than $2^{2(a_1-1)+\ldots+2(a_t-1)} \leq 2^{2s}$ ways. Let $a = \sum_i a_i$ be the total number of nodes. Then it remains to place $s - a + t$ edges which can be done in at most $s^{2(s-a+t)}$ ways. The number of ways that the nodes can be chosen is at most $n^{a-t+1}2^{a-1} \binom{(4k)^cs}{t-1}$ by arguing in the following manner: For each component $C_i$ we can describe one node by referring to $b_i$ and which endpoint the node is at. Thus we can describe $t-1$ of the nodes in at most $2^{t-1} \binom{|Y'|}{t-1}$ ways, where $Y'$ was the set $Y$ before the addition of the last edge, so $|Y'| \leq (4k)^cs$. The $t-1$ nodes can be picked in at most $2^{2s}$ ways, since
there are at most $2s$ nodes in $C_1, \ldots, C_t$. The remaining $a - t + 1$ nodes can be chosen in no more than $n^{a - t + 1}$ ways. Assuming that $n$ is larger than a constant we know by Lemma 8 that the number of ways to choose the keys in $S$ (including the order in which they were added) is bounded by $s^{O(1)}m^{s - k}$. Hence for a fixed $a$ the total number of ways to choose $S$ is at most:

$$2^{4s} \cdot s^{2(s-a+t)} \cdot n^{a-t+1} \cdot 2^{t-1} \left( \frac{(4k)^c s}{t} \right) \cdot s^{O(1)}m^{s-k}$$

For each of the $s$ independent keys we fix $2$ hash values, so the probability that those values occur is at most $n^{-2s}$. Thus the total probability that we can find such $S$ for fixed values of $s, a, t$ is at most:

$$2^{4s} \cdot s^{2(s-a+t)+O(1)}n^{a-t+1-2s} \cdot 2^{t-1} \left( \frac{(4k)^c s}{t} \right) \cdot s^{O(1)}m^{s-k} \leq n2^{4s} \left( \frac{s^2}{n} \right)^{s-a+t} \cdot s^{O(1)} \left( \frac{e(4k)^{c}s}{t-1} \right)^{t-1} \left( \frac{m}{n} \right)^s m^{-k} \leq ns^{O(1)} \left( \frac{2^5 e(4k)^{c}m}{n} \right)^s m^{-k} \leq s^{O(1)}m^{-k} \leq n(\lg n)^{O(1)}m^{-k}$$

Since there are at most $(2k \lg n)^3 = (\lg n)^{O(1)}$ ways to choose $s, a, t$ we can bound the probability by a union bound and get $n(\lg n)^{O(1)}m^{-k} = O(n^{-k+2})$.

**The algorithm stops because** $|S| \geq k \lg n$: Let $s, a, t$ have the same meaning as before. In the same way we can show (without using Lemma 8) that the number of ways to choose $S$ is bounded by

$$ns^{O(1)} \left( \frac{s^2}{n} \right)^{s-a+t} \left( \frac{2^5 e(4k)^{c}m}{n} \right)^s \leq ns^{O(1)}2^{-s}$$

Since $s = [k \lg n]$ we know that $2^{-s} \leq n^{-k}$ and a union bound over all choices of $a, t$ will suffice.

Along the same lines we can show that $s - a + t \leq k$ with probability $1 - O(n^{-k+2})$. The idea here is that we need to place $s - a + t$ additional keys when the spanning trees are fixed. Such a key and placement can be chosen in at most $ms^2$ ways, but it happens with probability at most $1/n^2$ due to the independence of the keys.

Assume there exists a component with arboricity $\alpha \geq 2(k + 2)(4k)^c$ and choose a subgraph $H$ such that $|E(H)| \geq \alpha(|V(H)| - 1)$. Consider the algorithm constructing $S$ restricted to $H$. If the algorithm is not stopped early we know that $Y$ contains exactly the edges of $H$, so $|Y| \geq |V(H)| \cdot (k + 2)(4k)^c$ and thus $|S| \geq (k + 2)|V(A)|$. This implies that $s - a + t \geq (k + 1)|V(A)| \geq k + 1$, i.e. every component has arboricity $\leq 2(k + 2)(4k)^c$ with probability $1 - O(n^{-k+2})$.

From the analysis above we get that there exists no component with more than $(4k)^c k \lg n$ nodes with probability $1 - O(n^{-k+2})$. Combining this with
Lemma 6 we now conclude that with probability \(1 - O(n^{-k+2})\) the maximum load is upper bounded by:

\[
2(k + 2)(4k)^c \cdot \lg ((4k)^ck \lg n) = O(\lg \lg n)
\]

The proof of Theorem 8 is now straightforward, since we just need to apply Lemma 9 \(\frac{2^5(4[\gamma + 2])^c m}{n} = O(1)\) times and take a union bound.

### 6.2 Expected maximum load

This section is dedicated to proving Theorem 7. The main idea is to bound the probability that a big binomial tree appears in the hash graph using the results of Section 2. A crucial point of the proof is to consider a subtree of the binomial tree which is chosen such that the number of leaves are much larger than the number of internal nodes.

**Proof of Theorem 7.** First of all note that by Theorem 8, the probability that the maximum load is more than \(k_0 \cdot \lg \lg n\) is \(O(n^{-1})\) for some constant \(k_0 > 1\). Hence it suffices to prove that the probability that the maximum load is larger than \(\lg \lg n + r + 1\) is at most \(O((\lg \lg n)^{-1})\) for some constant \(r\) depending on \(m/n\) and \(c\).

**Observation 2.** If there exists a bin with load at least \(k + 1\) then either there is a component with more edges than nodes or the binomial tree \(B_k\) is a subgraph of the hash graph.

**Proof.** Assume no component has more edges than nodes. Then, removing at most one edge from each component yields a forest. One edge per component will at most increase the load by 1, so consider the remaining forest.

Consider now the order in which the keys are inserted, and use induction on this order. Define \(G_j\) to be the graph after the \(j\)th key is inserted. The induction hypothesis is that if a bin has load \(k\), then it is the root in a subtree which is \(B_k\). For \(G_0\) it is easy to see. Consider now the addition of the \(j\)th key and assume that the hypothesis holds. Assume that the added key corresponds to the edge \((u, v)\) and that the load of bin \(u\) increases to \(l\). Since there are no cycles, node \(G_j\) must have edges \((u, v_0), \ldots, (u, v_{l-1})\), and by the induction hypothesis \(v_0, \ldots, v_{l-1}\) are roots of disjoint binomial trees \(B_0, \ldots, B_{l-1}\), so \(u\) is the root of a \(B_l\).

If \(m(1 + \varepsilon) < n\) we know from [17, Thm. 1.2] that no component of the hash graph contains a double cycle with probability \(O(n^{-1/3})\). Looking into the proof we see that there doesn’t exist a double cycle consisting of at most \(s\) edges with probability \((O(m/n))^{s n^{-1/3}}\) even when \(m > n\). In the terminology of [17] \(\lg(n/m)\) bits per edge is saved in the encoding of the hash-values.

\(^4\)A binomial tree \(B_i\) is a root which children are exactly the roots of binomial trees \(B_0, \ldots, B_{i-1}\), with \(B_0\) being a single node.
when \( \lg(n/m) < 0 \) we add \( \lg(m/n) \) extra bits in the encoding, i.e. that the bound on the probability is multiplied with \( (O(m/n))^s \). This means that we only need to bound the probability that there exists a binomial tree \( B_k, k = [\lg \lg n + r] \), because any bin with load \( k + 1 \) will either imply the existence of \( B_k \) in the hash graph or the existence of a double cycle consisting of \( \leq 4k = O(\lg \lg n) \) edges, and the latter happens with probability \( (\lg n)^{O(1)}n^{-1/3} = O(n^{-1/4}) \).

Say that the hash graph contains a binomial tree \( B_k \). Consider the subtree \( T_{k,d} \) defined by removing the children of all nodes that have less than \( d \) children, where \( d \leq k \) is some constant to be defined (see Figure 7). Note that \( T_{k,d} \) has \( (d + 1)2^{k-d} - 1 \) edges. We construct the ordered set \( S \) by traversing \( T_{k,d} \) in the following way: Order the edges in increasing distance from the root and on each level from left to right. Traverse the edges in this order. A given edge is added to the ordered set \( S \) if the following two requirements are fulfilled:

- After the edge is added \( S \) corresponds to a connected subgraph of \( T_{k,d} \).
- The key corresponding to the edge is independent of all the keys corresponding to the edges in \( S \).

A visualization of the set \( S \) can be seen in Figure 7. We will think of \( S \) as a set of edges, but also as a set of independent keys. The idea is to bound the probability that we could find such a set \( S \). We will split the proof into four cases depending on \( S \), and each will end with a ♦.

**Case 1:** \( s := |S| = (d + 1)2^{k-d} - 1 \): In this case every edge of the tree is independent, and there are at most \( m^s \) different ways to choose the ordered set \( S \). Note that there are \( 2^{k-d} \) groups of \( d \) leaves which have the same parent.
The set $S$ corresponds to the same subgraph of the hash graph regardless of the ordering of these leaves. Since we only want to bound the probability that we can find such $S$, we can thus chose the edges of $S$ in at most $m^s \left( \frac{1}{d!} \right)^{2k-d}$ ways. For a given choice of $S$ there are $s - 1$ equations $h_k(x) = h_k(y)$ which must be fulfilled where $k \in \{1, 2\}$ and $x, y$ are keys in $S$. Since the keys in $S$ are independent, the probability that this happens for a given $S$ is at most $2n^{-(s-1)}$. By a union bound on all the choices of $S$ the probability that such an $S$ exists is at most:

$$m^s \left( \frac{1}{d!} \right)^{2k-d} (2n^{-(s-1)}) \leq 2m^s \left( \frac{1}{\sqrt{d!}} \right)^{2k-d(d+1)} n^{-(s-1)} \leq 2n \cdot \left( \frac{m}{n^{\frac{s+1}{\sqrt{d!}}}!} \right)^s$$

We assume that $d$ and $r$ are chosen such that $\frac{m}{n^{\frac{s+1}{\sqrt{d!}}}!} < \frac{1}{2}$ and $r \geq d + 1$. Then $s \geq 2 \lg n$ and the probability is bounded by $2n^{-1}$. ◦

**Case 2: All the edges incident to the root lie in $S$:** Let $S'$ be defined in a similar manner as $S$: Order the edges in increasing distance from the root and on each level from left to right as before. Traverse the edges in this order, and add the edges to $S'$ if the corresponding key is independent of the keys in $S'$. However, stop this traversal the first time a dependent key occurs. In this way $S'$ will be an ordered subset of $S$ and the tree-structure will only depend on $s' = \mid S' \mid$. Fix this value $s'$. Since there is a key which is dependent on the keys in $S'$ there are at most $s^{O(1)} m^{s'-1}$ ways to choose $S'$ by Lemma 3 assuming that $s'^c \leq \frac{1}{2} m$, i.e. assuming that $n$ is larger than some constant depending on $c$.

Every internal node of $T_{k,d}$ has exactly $d$ children that are leaves. Therefore, there can be at most one node in $S'$ having less than $d$ children that are leaves and belong to $S'$. Let $v_1, \ldots, v_l$ denote the internal nodes in $S'$, where $l$ is the number of internal nodes. Let $w_i$ denote the number of children of $v_i$ that are leaves. Similar to case 1, the structure of $S'$ is independent of the order of the leaves with the same parent. Therefore $S'$ can be chosen in at most $s^{O(1)} m^{s'-1} \prod_{i=1}^{l} \frac{1}{w_i}$ ways. Since $w_i \geq \left( \frac{m}{n^{\frac{s+1}{\sqrt{d!}}}!} \right)^{\frac{1}{w_i}}$ we see that:

$$\prod_{i=1}^{l} \frac{1}{w_i} \leq \prod_{i=1}^{l} \left( \frac{e}{w_i} \right)^{w_i}$$

Letting $w = \sum_{i=1}^{l} w_i$ the concavity of $x \to x \log(e/x)$ combined with Jensen’s inequality yields:

$$\prod_{i=1}^{l} \left( \frac{e}{w_i} \right)^{w_i} \leq \left( \frac{le}{w} \right)^{w}$$

At most one of the $w_i$’s can be smaller than $d$, so wlog. assume that $w_1, \ldots, w_{l-1} \geq d$. The total number of nodes must be at least $l + d(l - 1)$, i.e.
\[ s' \geq l + d(l - 1) \text{ giving } l \leq \frac{s' + d}{d + 1}. \] Since \( l + w = s' \) we see that:

\[
\frac{l}{w} \leq \frac{s' + d}{d + 1} = \frac{1}{d} \cdot \frac{s' + d}{s' - 1} \leq \frac{2}{d}
\]

Where the last inequality holds assuming that \( n \) (and hence \( s' \geq \lg \lg n \)) is larger than a constant. Since \( w \geq (s - 1) \frac{d}{d + 1} \) we see that:

\[
\prod_{i=1}^{l} \frac{1}{w_i!} \leq \left( \frac{2e}{d} \right)^{\frac{d}{d + 1}}^{s' - 1}
\]

Assume that \( d \) is chosen such that \( \left( \frac{2e}{d} \right)^{\frac{d}{d + 1}} \leq \frac{n}{2m} \). The number of cases that we need to consider is then at most:

\[
s^{O(1)} m^{s' - 1} \prod_{i=1}^{l} \frac{1}{w_i!} \leq s^{O(1)} \left( \frac{m}{2} \right)^{s' - 1}
\]

Since \( S' \) is a tree there are \( s' - 1 \) equalities on the form \( h_k(x) = h_k(y) \) where \( k \in \{1, 2\}, x, y \in S' \) that must be satisfied if \( S' \) occurs. Since we know the tree structure from knowing \( s' \) there are at most two ways two choose these equalities. This means that the probability that a specific \( S' \) occurs is bounded by \( 2n^{-(s'-1)} \). For a fixed \( |S'| = s' \) the probability that there exists \( S' \) with \( s' \) elements is therefore bounded by:

\[
2s^{O(1)} \left( \frac{m}{2} \right)^{s' - 1} n^{-(s'-1)} = 2s^{O(1)} 2^{-s'+1}
\]

A union bound over all \( s' \geq \lg \lg n \) now yields the desired upper bound:

\[
\sum_{s' \geq \lg \lg n} 2s^{O(1)} 2^{-s'+1} \leq 2^{-\lg \lg n + 3} \sum_{k \geq 1} (k + [\lg \lg n] - 1)^{O(1)} 2^{-k}
\]

\[
\leq \frac{8}{\lg n} [\lg \lg n]^{O(1)} \sum_{k \geq 1} k^{O(1)} 2^{-k}
\]

\[
= \frac{(\lg \lg n)^{O(1)}}{\lg n}
\]

**Case 3: Not all, but at least \((\lg \lg n)/2\) edges incident to the root lie in \( S\):** Let \( S' \subseteq S \) be the set of independent keys adjacent to the root, and set \( s' = |S'|. \) By Lemma 3, \( S' \) can be chosen in no more than \( s^{O(1)} m^{s' - 1} s'! \) ways since there must exist a key (corresponding to an edge incident to the root) which is dependent on the keys in \( S' \) and the order of the keys are irrelevant. Since all the keys in \( S' \) are independent, the probability that \( h_0(x) \) or \( h_1(x) \) are the same for all the keys \( x \in S' \) is at most \( 2n^{-(s'-1)} \). So the probability that such a \( S' \) can be found is at most:

\[
\frac{(s^{O(1)} m^{s' - 1})}{s'!} (2n^{-(s'-1)}) = 2s^{O(1)} \left( \frac{m}{n} \right)^{s' - 1} \leq 2s^{O(1)} \left( \frac{mc}{ns'} \right)^{s' - 1} = O((\lg \lg n)^{-1})
\]

\[
\diamondsuit
\]
Case 4: There are less than \((\lg \lg n)/2\) edges incident to the root in \(S\): Let \(S' \subseteq S\) be the set of keys corresponding to the edges from \(S\) incident to the root and let \(s' = |S'|\). Since the other keys incident to the root must be dependent on the keys from \(S'\), Lemma 4 states that \(S'\) can be chosen in at most \(s'^{O(1)}m^{3/2}\) ways. Since all the keys in \(S'\) are independent the probability that \(h_0(x)\) or \(h_1(x)\) are the same for all the keys \(x \in S'\) is at most \(2^{-n}\). Thus, the probability of such a set \(S'\) occurring is bounded by:

\[
s'^{O(1)}m^{3/2} \cdot 2^{-n} \leq s'^{O(1)}2^{-n - (s' - 1)/2} \leq s'^{O(1)}2^{-n - 1/2}
\]

Consider the case of distributing \(m\) balls into \(n\) bins. Note that the proof actually gives an expected maximum load of \(O(m/n) + \lg \lg n + O(1)\) if \(m/n = o((\lg n)/(\lg \lg n))\). However, this only matches the behaviour of truly random hash functions under the assumption that \(m = O(n)\).

The same techniques can be used to show that \(\Omega(m \log n)-\)independent hash functions yield a maximum load of \(O(m/n) + \lg \lg n + O(1)\) with high probability (this is essentially case 1 in the proof). This implies that \(\Omega(\lg n)-\)independence hashing is sufficient to give the same theoretical guarantees as truly random hash functions in the context of the power of two choices when \(m = O(n)\).

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy, *The space complexity of approximating the frequency moments*, Journal of Computer and System Sciences 58 (1999), no. 1, 209–223, See also STOC’96.

[2] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal, *Balanced allocations*, SIAM Journal on Computing 29 (1999), no. 1, 180–200, See also STOC’94.

[3] Yoram Bachrach and Ely Porat, *Sketching for big data recommender systems using fast pseudo-random fingerprints*, Proc. 40th International Colloquium on Automata, Languages and Programming (ICALP), 2013, pp. 459–471.

[4] Paolo Boldi, Marco Rosa, and Sebastiano Vigna, *Hyperanf: Approximating the neighbourhood function of very large graphs on a budget*, Proc. 20th WWW, ACM, 2011, pp. 625–634.

[5] Andrei Z. Broder, *On the resemblance and containment of documents*, Proc. Compression and Complexity of Sequences (SEQUENCES), 1997, pp. 21–29.
[6] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher, *Min-wise independent permutations*, Journal of Computer and System Sciences 60 (2000), no. 3, 630–659, See also STOC’98.

[7] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig, *Syntactic clustering of the web*, Computer Networks 29 (1997), 1157–1166.

[8] L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder, *Balls and bins: Smaller hash families and faster evaluation*, Proc. 52nd IEEE Symposium on Foundations of Computer Science (FOCS), 2011, pp. 599–608.

[9] Moses Charikar, Kevin Chen, and Martin Farach-Colton, *Finding frequent items in data streams*, Proc. 29th International Colloquium on Automata, Languages and Programming (ICALP), Springer-Verlag, 2002, pp. 693–703.

[10] Edith Cohen, *All-distances sketches, revisited: Hip estimators for massive graphs analysis*, Proc. 33rd ACM Symposium on Principles of Database Systems, ACM, 2014, pp. 88–99.

[11] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup, *The power of two choices with simple tabulation*, CoRR abs/1407.6846 (2014).

[12] Philippe Flajolet and G. Nigel Martin, *Probabilistic counting algorithms for data base applications*, Journal of Computer and System Sciences 31 (1985), no. 2, 182–209.

[13] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and et al., *Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm*, In Analysis of Algorithms (AOFA), 2007.

[14] Ping Li, Art B. Owen, and Cun-Hui Zhang, *One permutation hashing*, Advances in Neural Information Processing Systems, 2012, pp. 3122–3130.

[15] Michael Mitzenmacher, Andréa W. Richa, and Ramesh Sitaraman, *The power of two random choices: A survey of techniques and results*, Handbook of Randomized Computing: volume 1, 2001, pp. 255–312.

[16] Anna Pagh and Rasmus Pagh, *Uniform hashing in constant time and optimal space*, SIAM J. Comput. 38 (2008), no. 1, 85–96.

[17] Mihai Pătraşcu and Mikkel Thorup, *The power of simple tabulation-based hashing*, Journal of the ACM 59 (2012), no. 3, Article 14, Announced at STOC’11.

[18] Mihai Pătraşcu and Mikkel Thorup, *Twisted tabulation hashing*, Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA), 2013, pp. 209–228.
[19] Omer Reingold, Ron D. Rothblum, and Udi Wieder, *Pseudorandom graphs in data structures*, Proc. 41st International Colloquium on Automata, Languages and Programming (ICALP), 2014, pp. 943–954.

[20] Anshumali Shrivastava and Ping Li, *Densifying one permutation hashing via rotation for fast near neighbor search*, Proc. 31th International Conference on Machine Learning (ICML), 2014, pp. 557–565.

[21] Alan Siegel, *On universal classes of extremely random constant-time hash functions*, SIAM Journal on Computing 33 (2004), no. 3, 505–543, See also FOCS’89.

[22] Mikkel Thorup, *Bottom-k and priority sampling, set similarity and subset sums with minimal independence*, Proc. 45th ACM Symposium on Theory of Computing (STOC), 2013.

[23] _____, *Simple tabulation, fast expanders, double tabulation, and high independence*, FOCS, 2013, pp. 90–99.

[24] Mikkel Thorup and Yin Zhang, *Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation*, SIAM Journal on Computing 41 (2012), no. 2, 293–331, Announced at SODA’04 and ALENEX’10.

[25] Mark N. Wegman and Larry Carter, *New classes and applications of hash functions*, Journal of Computer and System Sciences 22 (1981), no. 3, 265–279, See also FOCS’79.

[26] Albert Lindsey Zobrist, *A new hashing method with application for game playing*, Tech. Report 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.