


Power of d Choices with Simple Tabulation


Anders Aamand¹

BARC, University of Copenhagen, Universitetsparken 1, Copenhagen, Denmark.
aa@di.ku.dk

 <https://orcid.org/0000-0002-0402-0514>


Mathias Bæk Tejs Knudsen

University of Copenhagen and Supwiz, Copenhagen, Denmark.
mathias@tejs.dk

 <https://orcid.org/0000-0001-5308-9609>

Mikkel Thorup¹

BARC, University of Copenhagen, Universitetsparken 1, Copenhagen, Denmark.
mikkel2thorup@gmail.com

 <https://orcid.org/0000-0001-5237-1709>

Abstract

We consider the classic d -choice paradigm of Azar et al. [STOC'94] in which m balls are put into n bins sequentially as follows: For each ball we are given a choice of d bins chosen according to d hash functions and the ball is placed in the least loaded of these bins, breaking ties arbitrarily. The interest is in the number of balls in the fullest bin after all balls have been placed.

In this paper we suppose that the d hash functions are simple tabulation hash functions which are easy to implement and can be evaluated in constant time. Generalising a result by Dahlgaard et al. [SODA'16] we show that for an arbitrary constant $d \geq 2$ the expected maximum load is at most $\frac{\lg \lg n}{\lg d} + O(1)$. We further show that by using a simple tie-breaking algorithm introduced by Vöcking [J.ACM'03] the expected maximum load is reduced to $\frac{\lg \lg n}{d \lg \varphi_d} + O(1)$ where φ_d is the rate of growth of the d -ary Fibonacci numbers. Both of these expected bounds match those known from the fully random setting.

The analysis by Dahlgaard et al. relies on a proof by Pătraşcu and Thorup [J.ACM'11] concerning the use of simple tabulation for cuckoo hashing. We require a generalisation to $d > 2$ hash functions, but the original proof is an 8-page tour de force of ad-hoc arguments that do not appear to generalise. Our main technical contribution is a shorter, simpler and more accessible proof of the result by Pătraşcu and Thorup, where the relevant parts generalise nicely to the analysis of d choices.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization, Mathematics of computing → Random graphs, Mathematics of computing → Probabilistic algorithms, Theory of computation → Online algorithms, Theory of computation → Data structures design and analysis, Theory of computation → Bloom filters and hashing

Keywords and phrases Hashing, Load Balancing, Balls and Bins, Simple Tabulation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.5

Related Version A full version of the paper is available at [1], <https://arxiv.org/abs/1804.09684>.

¹ Research supported by Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research and by his Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.



© Anders Aamand, Mathias B. T. Knudsen, and Mikkel Thorup;
licensed under Creative Commons License CC-BY

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 5; pp. 5:1–5:14



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

Suppose that we are to place $m = O(n)$ balls sequentially into n bins. If the positions of the balls are chosen independently and uniformly at random it is well-known that the maximum load of any bin is² $\Theta(\log n / \log \log n)$ whp (i.e. with probability $1 - O(n^{-\gamma})$ for arbitrarily large fixed γ). See for example [10] for a precise analysis.

Another allocation scheme is the **d -choice paradigm** (also called the d -choice balanced allocation scheme) first studied by Azar et al. [2]: The balls are inserted sequentially by for each ball choosing d bins, according to d hash functions h_1, \dots, h_d and placing the ball in the one of these d bins with the least load, breaking ties arbitrarily. Azar et al. [2] showed that using independent and fully random hash functions the maximum load surprisingly drops to at most $\frac{\log \log n}{\log d} + O(1)$ whp. This result triggered an extensive study of this and related types of load balancing schemes. Currently the paper by Azar et al. has more than 700 citations by theoreticians and practitioners alike. The reader is referred to the text book [13] or the recent survey [21] for thorough discussions. Applications are numerous and are surveyed in [11, 12].

An interesting variant was introduced by Vöcking [20]. Here the bins are divided into d groups each of size $g = n/d$ and for each ball we choose a single bin from each group. The balls are inserted using the d -choice paradigm but in case of ties we always choose the leftmost of the relevant bins i.e. the one in the group of the smallest index. Vöcking proved that in this case the maximum load drops further to $\frac{\log \log n}{d \log \varphi_d} + O(1)$ whp.

In this paper we study the use of simple tabulation hashing in the load balancing schemes by Azar et al. and by Vöcking.

1.1 Simple tabulation hashing

Recall that a hash function h is a map from a key universe U to a range R chosen with respect to some probability distribution on R^U . If the distribution is uniform we say that h is fully random but we may impose any probability distribution on R^U .

Simple tabulation hashing was first introduced by Zobrist [23]. In simple tabulation hashing $U = [u] = \{0, 1, \dots, u - 1\}$ and $R = [2^r]$ for some r . We identify R with the \mathbb{Z}_2 -vector space $(\mathbb{Z}_2)^r$. The keys $x \in U$ are viewed as vectors consisting of $c > 1$ characters $x = (x[0], \dots, x[c - 1])$ with each $x[i] \in \Sigma \stackrel{\text{def}}{=} [u^{1/c}]$. We always assume that $c = O(1)$. The simple tabulation hash function h is defined by

$$h(x) = \bigoplus_{i=0}^{c-1} h_i(x[i])$$

where $h_0, \dots, h_{c-1} : \Sigma \rightarrow R$ are chosen independently and uniformly at random from R^Σ . Here \oplus denotes the addition in R which can in turn be interpreted as the bit-wise XOR of the elements $h_i(x[i])$ when viewed as bit-strings of length r .

Simple tabulation is trivial to implement, and very efficient as the character tables h_0, \dots, h_{c-1} fit in fast cache. Pătraşcu and Thorup [15] considered the hashing of 32-bit keys divided into 4 8-bit characters, and found it to be as fast as two 64-bit multiplications. On computers with larger cache, it may be faster to use 16-bit characters. We note that the c character table lookups can be done in parallel and that character tables are never changed once initialised.

² All logarithms in this paper are binary.

In the d -choice paradigm, it is very convenient that all the output bits of simple tabulation are completely independent (the j th bit of $h(x)$ is the XOR of the j th bit of each $h_i(x[i])$). Using (dr) -bit hash values, can therefore be viewed as using d independent r -bit hash values, and the d choices can thus be computed using a single simple tabulation hash function and therefore only c lookups.

1.2 Main results

We will study the maximum load when the elements of a fixed set $X \subset U$ with $|X| = m$ are distributed into d groups of bins G_1, \dots, G_d each of size $g = n/d$ using the d -choice paradigm with independent simple tabulation hash functions $h_1, \dots, h_d : U \rightarrow [n/d]$. The d choices thus consist of a single bin from each group as in the scheme by Vöcking but for $x \in X$ we may identify $h_i(x)$ with $(h_i(x), i) \in [n/d] \times [d]$ and thus think of all h_i as mapping to the same set of bins like in the scheme by Azar et al.

Dahlgaard et al. [7] analysed the case $d = 2$ proving that if $m = O(n)$ balls are distributed into two tables each consisting of $n/2$ bins according to the two choice paradigm using two independently chosen simple tabulation hash functions, the maximum load of any bin is $O(\log \log n)$ whp. For $k = O(1)$ they further provided an example where the maximum load is at least $\lfloor k^{c-1}/2 \rfloor \log \log n - O(1)$ with probability $\Omega(n^{-2(k-1)(c-1)})$. Their example generalises to arbitrary fixed $d \geq 2$ so we cannot hope for a maximum load of $(1+o(1)) \frac{\log \log n}{\log d}$ or even $100 \times \log \log n$ whp when d is constant. However, as we show in the full version of this paper [1], their result implies that even with $d = O(1)$ choices the maximum load is $O(\log \log n)$ whp.

Dahlgaard et al. also proved that the expected maximum load is at most $\log \log n + O(1)$ when $d = 2$. We prove the following result which generalises this to arbitrary $d = O(1)$.

► **Theorem 1.** *Let $d > 1$ be a fixed constant. Assume $m = O(n)$ balls are distributed into d tables each of size n/d according to the d -choice paradigm using d independent simple tabulation hash functions $h_1, \dots, h_d : U \rightarrow [n/d]$. Then the expected maximum load is at most $\frac{\log \log n}{\log d} + O(1)$.*

When in the d -choice paradigm we sometimes encounter ties when placing a ball — several bins among the d choices may have the same minimum load. As observed by Vöcking [20] the choice of tie breaking algorithm is of subtle importance to the maximum load. In the fully random setting, he showed that if we use the **Always-Go-Left algorithm** which in case of ties places the ball in the leftmost of the relevant bins, i.e. in the bin in the group of the smallest index, the maximum load drops to $\frac{\log \log n}{d \log \varphi_d} + O(1)$ whp. Here φ_d is the *unique* positive real solution to the equation $x^d = x^{d-1} + \dots + x + 1$. We prove that his result holds in expectation when using simple tabulation hashing.

► **Theorem 2.** *Suppose that we in the setting of Theorem 1 use the Always-Go-Left algorithm for tie-breaking. Then the expected maximum load of any bin is at most $\frac{\log \log n}{d \log \varphi_d} + O(1)$.*

Note that φ_d is the rate of growth of the so called d -ary Fibonacci numbers for example defined by $F_d(k) = 0$ for $k \leq 0$, $F_d(1) = 1$ and finally $F_d(k) = F_d(k-1) + \dots + F_d(k-d)$ when $k > 1$. With this definition we can write $\varphi_d = \lim_{k \rightarrow \infty} \sqrt[k]{F_d(k)}$. It is easy to check that $(\varphi_d)_{d>1}$ is an increasing sequence converging to 2.

1.3 Technical contributions

In proving Theorem 1 we would ideally like to follow the approach by Dahlgaard et al. [7] for the case $d = 2$ as close as possible. They show that if some bin gets load $k + 1$ then

either the hash graph (informally, the d -uniform hypergraph with an edge $\{h_1(x), \dots, h_d(x)\}$ for each $x \in X$) contains a subgraph of size $O(k)$ with more edges than nodes or a certain kind of “witness tree” T_k . They then bound the probability that either of these events occur when $k = \log \log n + r$ for some sufficiently large constant r . Putting $k = \frac{\log \log n}{\log d} + r$ for a sufficiently large constant r we similarly have three tasks:

- (1) Define the d -ary witness trees and argue that if some bin gets load $k + 1$ then either **(A)**: the hash graph contains a such, or **(B)**: it contains a subgraph $G = (V, E)$ of size $O(k)$ with $|V| \leq (d - 1)|E| - 1$.
- (2) Bound the probability of **(A)**.
- (3) Bound the probability of **(B)**.

Step (1) and (2) require intricate arguments but the techniques are reminiscent to those used by Dahlgaard et al. in [7] and it is not surprising that their arguments generalise to our setting. Due to space limitations this part of our analysis can be found in the full version of this paper [1].

Our main technical contribution is our work on step (3) as we now describe. Dealing with step (3) in the case $d = 2$ Dahlgaard et al. used the proof by Pătraşcu and Thorup [15] of the result below concerning the use of simple tabulation for cuckoo hashing³.

► **Theorem 3** (Pătraşcu and Thorup [15]). *Fix $\varepsilon > 0$. Let $X \subset U$ be any set of m keys. Let n be such that $n > 2(1 + \varepsilon)m$. With probability $1 - O(n^{-1/3})$ the keys of X can be placed in two tables of size $n/2$ with cuckoo hashing using two independent simple tabulation hash functions h_0 and h_1 .*

Unfortunately for us, the original proof of Theorem 3 consists of 8 pages of intricate ad-hoc arguments that do not seem to generalise to the d -choice setting. Thus we have had to develop an alternative technique for dealing with step (3) As an extra reward this technique gives a new proof of Theorem 3 which is shorter, simpler and more readable and we believe it to be our main contribution and of independent interest⁴.

1.4 Alternatives

We have shown that balanced allocation with d choices with simple tabulation gives the same expected maximum load as with fully-random hashing. Simple tabulation uses c lookups in tables of size $u^{1/c}$ and $c - 1$ bit-wise XOR. The experiments from [15], with $u = 2^{32}$ and $c = 4$, indicate this to be about as fast as two multiplications.

Before comparing with alternative hash functions, we note that we may assume that $u \leq n^2$. If u is larger, we can first apply a universal hash function [3] from $[u]$ to $[n^2]$. This yields an expected number of $\binom{n}{2}/n^2 < 1/2$ collisions. We can now apply *any* hash function, e.g., simple tabulation, to the reduced keys in $[n^2]$. Each of the duplicate keys can increase the maximum load by at most one, so the expected maximum load increases by at most $1/2$. If $u = 2^w$, we can use the extremely simple universal hash function from [8], multiplying the key by a random odd w -bit number and performing a right-shift.

Looking for alternative hash functions, it can be checked that $O(\log n)$ -independence suffices to get the same maximum load bounds as with full randomness even with high

³ Recall that in cuckoo hashing, as introduced by Pagh and Rodler [14], we are in the 2-choice paradigm but we require that no two balls collide. However, we are allowed to rearrange the balls at any point and so the feasibility does only depend on the choices of the balls.

⁴ We mention in passing that Theorem 3 is best possible: There exists a set X of m keys such that with probability $\Omega(n^{-1/3})$ cuckoo hashing is forced to rehash (see [15]).

probability. High independence hash functions were pioneered by Siegel [17] and the most efficient construction is the double tabulation of Thorup [18]. It gives independence $u^{\Omega(1/c^2)}$ using space $O(cu^{1/c})$ in time $O(c)$. With c a constant this would suffice for our purposes. However, looking into the constants suggested in [18], with 16-bit characters for 32-bit keys, we have 11 times as many character table lookups with double tabulation as with simple tabulation and we lose the same factor in space, so this is not nearly as efficient.

Another approach was given by Woelfel [22] using the hash functions he earlier developed with Dietzfelbinger [9]. He analysed Vöcking's Always-Go-Left algorithm, bounding the error probability that the maximum load exceeded $\frac{\log \log n}{d \log \varphi_d} + O(1)$. Slightly simplified and translated to match our notation, using $d + 1$ k -independent hash functions and d lookups in tables of size $n^{2/c}$, the error probability is $n^{1+o(1)-k/c}$. Recall that we may assume $n^{2/c} \geq u^{1/c}$, so this matches the space of simple tabulation with c characters. With, say, $c = 4$, he needs 5-independent hashing to get any non-trivial bound, but the fastest 5-independent hashing is the tabulation scheme of Thorup and Zhang [19], which according to the experiments in [15] is at least twice as slow as simple tabulation, and much more complicated to implement.

A final alternative is to compromise with the constant evaluation time. Reingold et al. [16] have shown that using the hash functions from [4] yields a maximum load of $O(\log \log n)$ whp. The functions use $O(\log n \log \log n)$ random bits and can be evaluated in time $O((\log \log n)^2)$. Very recently Chen [5] used a refinement of the hash family from [4] giving a maximum load of at most $\frac{\log \log n}{\log d} + O(1)$ whp and $\frac{\log \log n}{d \log \varphi_d} + O(1)$ whp using the Always-Go-Left algorithm. His functions require $O(\log n \log \log n)$ random bits and can be evaluated in time $O((\log \log n)^4)$. We are not so concerned with the number of random bits. Our main interest in simple tabulation is in the constant evaluation time with a very low constant.

1.5 Structure of the paper

In Section 2 we provide a few preliminaries for the proofs of our main results. In Section 3 we deal with step (3) described under *Technical contributions*. To provide some intuition we first provide the new proof of Theorem 3. Finally, we show how to proceed for general d . For step (1) and (2) as well as the final deduction of Theorem 1 and Theorem 2 the reader is referred to the full version of this paper [1].

2 Preliminaries

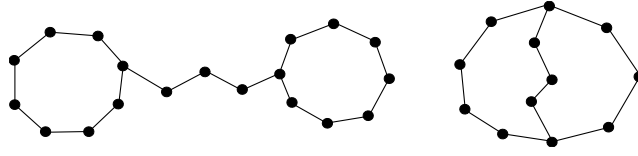
First, recall the definition of a hypergraph:

► **Definition 4.** A **hypergraph** is a pair $G = (V, E)$ where V is a set and E is a multiset consisting of elements from $\mathcal{P}(V)$. The elements of V are called **vertices** and the elements of E are called **edges**. We say that G is **d -uniform** if $|e| = d$ for all $e \in E$.

When using the d -choice paradigm to distribute a set of keys X there is a natural d -uniform hypergraph associated with the keys of X .

► **Definition 5.** Given a set of keys $X \subset U$ the **hash graph** is the d -uniform hypergraph on $[n/d] \times [d]$ with an edge $\{(h_1(x), 1), \dots, (h_d(x), d)\}$ for each $x \in X$.

When working with the hash graph we will hardly ever distinguish between a key x and the corresponding edge, since it is tedious to write $\{(h_i(x), i)\}_{1 \leq i \leq d}$. Statements such as “ $P = (x_1, \dots, x_t)$ is a path” or “The keys x_1 and x_2 are adjacent in the hash graph” are examples of this abuse of notation.



■ **Figure 1** Double cycles - the minimal obstructions for cuckoo hashing.

Now we discuss the independence of simple tabulation. First recall that a **position character** is an element $(j, \alpha) \in [c] \times \Sigma$. With this definition a key $x \in U$ can be viewed as the set of position characters $\{(i, x[i])\}_{i=0}^{c-1}$ but it is sensible to define $h(S) = \bigoplus_{i=1}^k h_{j_i}(\alpha_i)$ for any set $S = \{(j_1, \alpha_1), \dots, (j_k, \alpha_k)\}$ of position characters.

In the classical notion of independence of Carter and Wegman [3] simple tabulation is not even 4-independent. In fact, the keys $(a_0, b_0), (a_0, b_1), (a_1, b_0)$ and (a_1, b_1) are dependent, the issue being that each position character appears an even number of times and so the bitwise XOR of the hash values will be the zero string. As proved by Thorup and Zhang [19] this property in a sense characterises dependence of keys.

► **Lemma 6** (Thorup and Zhang [19]). *The keys $x_1, \dots, x_k \in U$ are dependent if and only if there exists a non-empty subset $I \subset \{1, \dots, k\}$ such that each position character in $(x_i)_{i \in I}$ appears an even number of times. In this case we have that $\bigoplus_{i \in I} h(x_i) = 0$.*

When each position character appears an even number of times in $(x_i)_{i \in I}$ we will write $\bigoplus_{i \in I} x_i = \emptyset$ which is natural when we think of a key as a set of position characters and \oplus as the symmetric difference. As shown by Dahlggaard et al. [6] the characterisation in Lemma 6 can be used to bound the independence of simple tabulation.

► **Lemma 7** (Dahlggaard et al. [6]). *Let $A_1, \dots, A_{2t} \subset U$. The number of $2t$ -tuples $(x_1, \dots, x_{2t}) \in A_1 \times \dots \times A_{2t}$ such that $x_1 \oplus \dots \oplus x_{2t} = \emptyset$ is at most⁵ $((2t - 1)!!)^c \prod_{i=1}^{2t} \sqrt{|A_i|}$.*

This lemma will be of extreme importance to us. In the full version of this paper [1] proofs of both Lemma 6 and Lemma 7 can be found.

3 Cuckoo hashing and generalisations

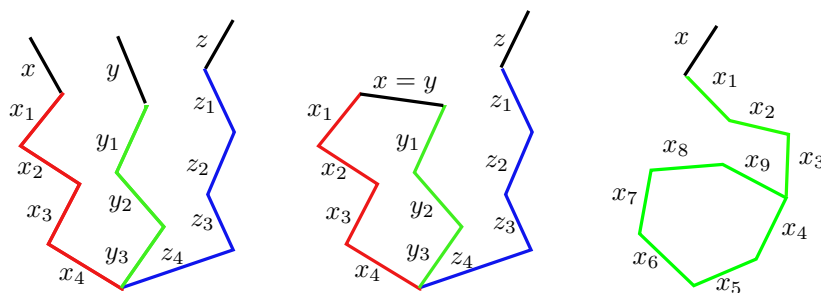
The following result is a key ingredient in the proofs of Theorem 1 and Theorem 2.

► **Theorem 8.** *Suppose that we are in the setting of Theorem 1 i.e. $d > 1$ is a fixed constant, $X \subset U$ with $|X| = m = O(n)$ and $h_1, \dots, h_d : U \rightarrow [n/d]$ are independent simple tabulation hash functions. The probability that the hash graph contains a subgraph $G = (V, E)$ of size $|E| = O(\log \log n)$ with $|V| \leq (d - 1)|E| - 1$ is at most $n^{-1/3+o(1)}$.*

Before giving the full proof however we provide the new proof of Theorem 3 which is more readable and illustrates nearly all the main ideas.

Proof of Theorem 3. It is well known that cuckoo hashing is possible if and only if the hash graph contains no subgraph with more edges than nodes. A minimal such graph is called a **double cycle** and consists of two cycles connected by a path or two vertices connected by

⁵ Recall the double factorial notation: If a is a positive integer we write $a!!$ for the product of all the positive integers between 1 and a that have the same parity as a .



■ **Figure 2** Non-black edges: Two tridents and a lasso. Black edges: Keys that are each dependent on the set of coloured keys.

three disjoint paths (see Figure 1). Hence, it suffices to bound the probability that the hash graph contains a double cycle by $O(n^{-1/3})$.

We denote by g the number of bins in each of the two groups. Thus in this setting $g = n/2 \geq (1 + \varepsilon)m$. First of all, we argue that we may assume that the hash graph contains no trail of length at least $\ell = \frac{4}{3} \frac{\log n}{\log(1+\varepsilon)}$ consisting of *independent*. Indeed, the keys of a such can be chosen in at most m^ℓ ways and since we require $\ell - 1$ equations of the form $h_i(x) = h_i(y)$, $i \in \{1, 2\}$ to be satisfied and since these events are independent the probability that the hash graph contains such a trail is by a union bound at most

$$\frac{2m^\ell}{g^{\ell-1}} \leq \frac{n}{(1 + \varepsilon)^\ell} = n^{-1/3}.$$

Now we return to the double cycles. Let A_ℓ denote the event that the hash graph contains a double cycle of size ℓ consisting of *independent* keys. The graph structure of a such can be chosen in $O(\ell^2)$ ways and the keys (including their positions) in at most m^ℓ ways. Since there are $\ell + 1$ equations of the form $h_i(x) = h_i(y)$, $i \in \{1, 2\}$ to be satisfied the probability that the hash graph contains a double cycle consisting of independent keys is at most

$$\sum_{\ell=3}^m \mathbb{P}(A_\ell) = O\left(\sum_{\ell=3}^m \ell^2 \frac{m^\ell}{g^{\ell+1}}\right) = O\left(\frac{1}{n} \sum_{\ell=3}^m \frac{\ell^2}{(1 + \varepsilon)^\ell}\right) = O(n^{-1}).$$

The argument above is the same as in the fully random setting. We now turn to the issue of dependencies in the double cycle starting with the following definition.

► **Definition 9.** We say that a graph is a **trident** if it consists of three paths P_1, P_2, P_3 of non-zero lengths meeting at a single vertex v . (see the non-black part of Figure 2).

We say that a graph is a **lasso** if it consists of a path that has one end attached to a cycle (see the non-black part of Figure 2).

We claim that in any double cycle D consisting of *dependent* keys we can find one of the following structures (see Figure 2):

- **S1:** A lasso L consisting of independent keys together with a key x not on L and incident to the degree 1 vertex of L such that x is dependent on the keys of L .
- **S2:** A trident T consisting of independent keys together with 3 (not necessarily distinct) keys x, y, z not on T but each dependent on the keys of T and incident to the 3 vertices of degree 1 on T

To see this suppose first that one of the cycles C of D consists of independent keys. In this case any maximal lasso of independent keys in D containing the edges of C is an S_1 .

On the other hand if all cycles contained in D consist of dependent keys we pick a vertex of D of degree at least 3 and 3 incident edges. These 3 edges form an independent trident (simple tabulation is 3-independent) and any maximal independent trident contained in D and containing these edges forms an S_2 .

Our final step is thus to show that the probability that these structures appear in the hash graph is $O(n^{-1/3})$

The lasso (S_1):

Since the edges of the lasso form an independent trail it by the initial observation suffices to bound the probability that the hash graph contains an S_1 of size ℓ for any $\ell = O(\log n)$.

Fix the size ℓ of the lasso. The number of ways to choose the graph structure of the lasso is $\ell - 2 < \ell$. Denote the set of independent keys of the lasso by $S = \{x_1, \dots, x_\ell\}$ and let x be the dependent key in S_1 . By Lemma 6 we may write $x = \bigoplus_{i \in I} x_i$ for some $I \subset \{1, \dots, \ell\}$. Fix the size $|I| = t \geq 3$ (which is necessarily odd). By Lemma 7 the number of ways to choose the keys of $(x_i)_{i \in I}$ (including their order) is at most $(t!)^c m^{(t+1)/2}$ and the number of ways to choose their positions in the lasso is $\binom{\ell}{t}$. The number of ways to choose the remaining keys of S is trivially bounded by $m^{\ell-t}$ and the probability that the choice of independent keys hash to the correct positions in the lasso is at most $2/g^\ell$. By a union bound the probability that the hash graph contains an S_1 for fixed values of ℓ and t is at most

$$\ell (t!)^c m^{(t+1)/2} m^{\ell-t} \binom{\ell}{t} \frac{2}{g^\ell}.$$

This is maximised for $t = 3$. In fact, when $\ell \leq m^{1/(c+2)}$ and $t \leq \ell - 2$ we have that

$$\frac{((t+2)!)^c m^{(t+3)/2} m^{\ell-t-2} \binom{\ell}{t+2}}{(t!)^c m^{(t+1)/2} m^{\ell-t} \binom{\ell}{t}} = \frac{(t+2)^c \binom{\ell-t}{2}}{m \binom{t+2}{2}} \leq \frac{\ell^{c+2}}{m} \leq 1.$$

Thus the probability that the hash graph contains an S_1 of size $O(\log n)$ is at most

$$\sum_{\ell=3}^{O(\log n)} \sum_{t=3}^{\ell} \ell 3^c \binom{\ell}{3} \frac{2m^{\ell-1}}{g^\ell} = O\left(\sum_{\ell=3}^{O(\log n)} \frac{\ell^5}{n(1+\varepsilon)^{\ell-1}}\right) = O(n^{-1}).$$

The trident (S_2):

Fix the size ℓ of the trident. The number of ways to choose the structure of the trident is bounded by ℓ^2 (once we choose the lengths of two of the paths the length of the third becomes fixed). Let $P_1 = (x_1, \dots, x_{t_1})$, $P_2 = (y_1, \dots, y_{t_2})$ and $P_3 = (z_1, \dots, z_{t_3})$ be the three paths of the trident meeting in $x_{t_1} \cap y_{t_2} \cap z_{t_3}$. As before we may assume that each has length $O(\log n)$. Let S denote the keys of the trident and enumerate $S = \{w_1, \dots, w_\ell\}$ in some order. Write $x = \bigoplus_{i \in I} w_i$, $y = \bigoplus_{j \in J} w_j$ and $z = \bigoplus_{k \in K} w_k$ for some $I, J, K \subset \{1, \dots, \ell\}$. By a proof almost identical to that given for the lasso we may assume that $|I| = |J| = |K| = 3$. Indeed, if for example $|I| \geq 5$ we by Lemma 7 save a factor of nearly m^2 when choosing the keys of S and this makes up for the fact that the trident contains no cycles and hence that the probability of a fixed set of independent keys hashing to it is a factor of g larger.

The next observation is that we may assume that $|I \cap J|, |J \cap K|, |K \cap I| \geq 2$. Again the argument is of the same flavour as the one given above. If for example $|I \cap J| = 1$ we by an application of Lemma 7 obtain that the number of ways to choose the keys of $(w_i)_{i \in I}$ is $O(m^2)$. Conditioned on this, the number of ways to choose the keys $(w_j)_{j \in J}$ is $O(m^{3/2})$ by

another application of Lemma 7 with one of the A_i 's a singleton. Thus we save a factor of $m^{3/2}$ when choosing the keys of S which will again suffice. The bound gets even better when $|I \cap J| = 0$ where we save a factor of m^2 .

Suppose now that x_1 is not a summand of $\bigoplus_{i \in I} w_i$. Write $x = w_a \oplus w_b \oplus w_c$ and let A be the event that the independent keys of S hash to the trident (with the equation involving x_1 and x_2 being $h_2(x_1) = h_2(x_2)$ without loss of generality). Then $\mathbb{P}(A) = \frac{1}{g^{\ell-1}}$. We observe that

$$\mathbb{P}(h_1(x) = h_1(x_1) \mid A) = \mathbb{P}(h_1(x_1) = h_1(w_a) \oplus h_1(w_b) \oplus h_1(w_c) \mid A) = g^{-1}$$

since A is a conjunction of events of the form $\{h_i(w) = h_i(w')\}$ none of them involving $h_1(x_1)$ ⁶. A union bound then gives that the probability that this can happen is at most

$$\sum_{\ell=3}^{O(\log n)} \ell^2 \binom{\ell}{3} (3!!)^c m^2 m^{\ell-3} \left(\frac{1}{g}\right)^\ell = O\left(\frac{1}{n} \sum_{\ell=3}^{\infty} \frac{\ell^5}{(1+\varepsilon)^{\ell-1}}\right) = O(n^{-1}).$$

Thus we may assume that x_1 is a summand of $\bigoplus_{i \in I} w_i$ and by similar arguments that y_1 is a summand of $\bigoplus_{j \in J} w_j$ and that z_1 is a summand of $\bigoplus_{k \in K} w_k$.

To complete the proof we need one final observation. We can define an equivalence relation on $X \times X$ by $(a, b) \sim (c, d)$ if $a \oplus b = c \oplus d$. Denote by $\mathcal{C} = \{C_1, \dots, C_r\}$ the set of equivalence classes. One of them, say C_1 , consists of the elements $(x, x)_{x \in X}$. We will say that the equivalence class C_i is **large** if $|C_i| \geq m^{2/3}$ and **small** otherwise. Note that

$$\sum_{i=1}^r |C_i|^2 = |\{(a, b, c, d) \in X^4 : a \oplus b \oplus c \oplus d = \emptyset\}| \leq 3^c m^2$$

by Lemma 7. In particular the number of large equivalence classes is $O(m^{2/3})$.

If h is a simple tabulation hash function we can well-define a map $\tilde{h} : \mathcal{C} \rightarrow R$ by $\tilde{h}(a, b) = h(a) \oplus h(b)$. Since the number of large equivalence classes is $O(m^{2/3})$ the probability that $\tilde{h}_i(C) = 0$ for some large $C \in \mathcal{C} \setminus \{C_1\}$ and some $i \in \{1, 2\}$ is $O(m^{2/3}/n) = O(n^{-1/3})$ and we may thus assume this does not happen.

In particular, we may assume that (x, x_1) , (y, y_1) and (z, z_1) each represent small equivalence classes as they are adjacent in the hash graph. Now suppose that y_1 is not a summand in $x = \bigoplus_{i \in I} w_i$. The number of ways to pick $(x_i)_{i \in I}$ is at most $3^c m^2$ by Lemma 7. By doing so we fix the equivalence class of (y, y_1) but not y_1 so conditioned on this the number of ways to pick $(y_j)_{j \in J}$ is at most $m^{2/3}$. The number of ways to choose the remaining keys is bounded by $m^{\ell-4}$ and a union bound gives that the probability of having such a trident is at most

$$\sum_{\ell=3}^{O(\log n)} \ell^2 3 \binom{\ell}{2} 3^c m^2 m^{2/3} m^{\ell-4} \left(\frac{1}{g}\right)^{\ell-1} = O\left(n^{-1/3} \sum_{\ell=3}^{\infty} \frac{\ell^4}{(1+\varepsilon)^{\ell-4/3}}\right) = O(n^{-1/3}),$$

which suffices.

We may thus assume that y_1 is a summand in $\bigoplus_{i \in I} w_i$ and by an identical argument that z_1 is a summand in $\bigoplus_{i \in I} w_i$ and hence $x = x_1 \oplus y_1 \oplus z_1$. But the same arguments apply to y and z reducing to the case when $x = y = z = x_1 \oplus y_1 \oplus z_1$ which is clearly impossible. \blacktriangleleft

⁶ If $x_1 = w_a$, say, we don't necessarily get the probability g^{-1} . In this case the probability is $\mathbb{P}(h_1(w_b) = h_1(w_c) \mid A)$ and the event $\{h(w_b) = h(w_c)\}$ might actually be included in A in which case the probability is 1. This can of course only happen if the keys w_b and w_c are *adjacent* in the trident so we could impose even further restrictions on the dependencies in S_2 .

3.1 Proving Theorem 8

Now we will explain how to prove Theorem 8 proceeding much like we did for Theorem 3. Let us say that a d -uniform hypergraph $G = (V, E)$ is **tight** if $|V| \leq (d-1)|E| - 1$. With this terminology Theorem 8 states that the probability that the hash graph contains a tight subgraph of size $O(\log \log n)$ is at most $n^{-1/3+o(1)}$. It clearly suffices to bound the probability of the existence of a *connected* tight subgraph of size $O(\log \log n)$.

We start with the following two lemmas. The counterparts in the proof of Theorem 3 are the bounds on the probability of respectively an independent double cycle and an independent lasso with a dependent key attached.

► **Lemma 10.** *Let A_1 denote the event that the hash graph contains a tight subgraph $G = (V, E)$ of size $O(\log \log n)$ consisting of independent keys. Then $\mathbb{P}(A_1) \leq n^{-1+o(1)}$.*

Proof. Let $\ell = |E|$ be fixed. The number of ways to choose the keys of E is trivially bounded by m^ℓ and the number of ways to choose the set of nodes V in the hash graph is $\binom{n}{(d-1)\ell-1}$. For such a choice of nodes let a_i denote the number of nodes of V in the i 'th group. The probability that one of the keys hash to V is then

$$\prod_{i=1}^d \frac{da_i}{n} \leq \left(\frac{a_1 + \dots + a_d}{n} \right)^d \leq \left(\frac{d\ell}{n} \right)^d.$$

By the independence of the keys and a union bound we thus have that

$$\mathbb{P}(A_1) \leq \sum_{\ell=2}^{O(\log \log n)} m^\ell \binom{n}{(d-1)\ell-1} \left(\frac{d\ell}{n} \right)^{d\ell} \leq \sum_{\ell=2}^{O(\log \log n)} \frac{1}{n} \left(\frac{m}{n} \right)^\ell (d\ell)^{d\ell} = n^{-1+o(1)},$$

as desired. ◀

► **Lemma 11.** *Let A_2 be the event that the hash graph contains a subgraph $G = (V, E)$ with $|V| \leq (d-1)|E|$ and $|E| = O(\log \log n)$ such that the keys of E are independent but such that there exists a key $y \notin E$ dependent on the keys of E . Then $\mathbb{P}(A_2) \leq n^{-1+o(1)}$.*

Proof. Let $|E| = \ell$ be fixed and write $E = \{x_1, \dots, x_\ell\}$. We want to bound the number of ways to choose the keys of E . By Lemma 6, $y = \bigoplus_{i \in I} x_i$ for some $I \subset \{1, \dots, \ell\}$ with $|I| = r$ for some odd $r \geq 3$. Let r be fixed for now. Using Lemma 7, we see that the number of ways to choose the keys of E is no more than $(r!!)^c m^{\frac{r+1}{2}} m^{\ell-r}$. For fixed ℓ and r the probability is thus bounded by

$$(r!!)^c m^{\ell - \frac{r+1}{2}} \binom{n}{\ell(d-1)} \left(\frac{d\ell}{n} \right)^{d\ell} = n^{-1+o(1)}$$

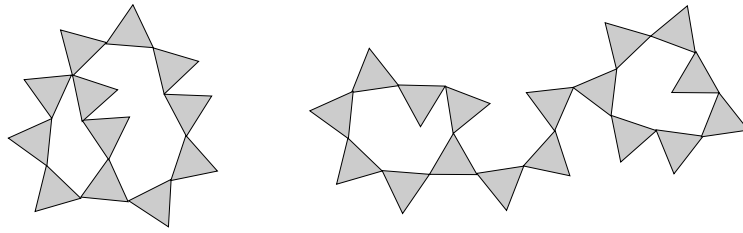
and a union bound over all $\ell = O(\log \log n)$ and $r \leq \ell$ suffices. ◀

We now generalise the notion of a double cycle starting with the following definition.

► **Definition 12.** Let $G = (V, E)$ be a d -uniform hypergraph. We say that a sequence of edges $P = (e_1, \dots, e_t)$ of G is a **path** if $|e_i \cap e_{i+1}| = 1$ for $1 \leq i \leq t-1$ and $e_i \cap e_j = \emptyset$ when $i < j-1$.

We say that $C = (e_1, \dots, e_t)$ is a **cycle** if $t \geq 3$, $|e_i \cap e_{i+1}| = 1$ for all $i \pmod{t}$ and $e_i \cap e_j = \emptyset$ when $i \not\equiv j \pm 1 \pmod{t}$.

Next comes the natural extension of the definition of double cycles to d -uniform hypergraphs.



■ **Figure 3** Double cycles in the case $d = 3$. The triangles represent edges of the graph and the corners represent the vertices.

► **Definition 13.** A d -uniform hypergraph G is called a **double cycle** if it has either of the following forms (see Figure 3).

- **D1:** It consists of two vertex disjoint cycles C_1 and C_2 connected by a path $P = (x_1, \dots, x_t)$ such that $|x_1 \cap V(C_1)| = |x_t \cap V(C_2)| = 1$ and $x_{i+1} \cap V(C_1) = x_i \cap V(C_2) = \emptyset$ for $1 \leq i \leq t - 1$. We also allow P to have zero length and $|V(C_1) \cap V(C_2)| = 1$.
- **D2:** It consist of a cycle C and a path $P = (x_1, \dots, x_t)$ of length $t \geq 2$ such that $|x_1 \cap V(C)| = |x_t \cap V(C)| = 1$ and $x_i \cap V(C) = \emptyset$ for $2 \leq i \leq t - 1$. We also allow $t = 1$ and $|x_1 \cap C| = 2$.

Note that a double cycle always has $|V| = (d - 1)|E| - 1$.

Now assume that the hash graph contains a connected tight subgraph $G = (V, E)$ of size $O(\log \log n)$ but that neither of the events of Lemma 10 and 11 has occurred. In particular no two edges e_1, e_2 of G has $|e_1 \cap e_2| \geq 2$ and no cycle consists of independent keys.

It is easy to check that under this assumption G contains at least two cycles. Now pick a cycle C_1 of least possible length. Since simple tabulation is 3-independent the cycle consists of at least 4 edges. If there exists an edge x not part of C_1 with $|x \cap V(C_1)| = 2$ we get a double cycle of type D_2 . If $|x \cap V(C_1)| \geq 3$ we can use x to obtain a shorter cycle than C_1 which is a contradiction⁷. Using this observation we see that if there is a cycle $C_2 \neq C_1$ such that $|V(C_1) \cap V(C_2)| \geq 2$ then we can find a D_2 in the hash graph. Thus we may assume that any cycle $C_2 \neq C_1$ satisfies $|V(C_2) \cap V(C_1)| \leq 1$.

Now pick a cycle C_2 different from C_1 of least possible length. As before we may argue that any edge x not part of C_2 satisfies that $|x \cap V(C_2)| \leq 1$. Picking a shortest path connecting C_1 and C_2 (possibly the length is zero) gives a double cycle of type D_1 .

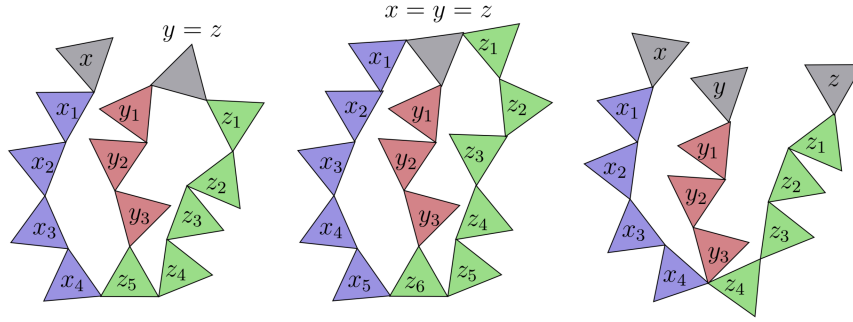
Next we define tridents (see the non-grey part of Figure 4).

► **Definition 14.** We call a d -uniform hypergraph T a **trident** if it consists of paths $P_1 = (x_1, \dots, x_{t_1})$, $P_2 = (y_1, \dots, y_{t_2})$ and $P_3 = (z_1, \dots, z_{t_3})$ of non-zero length such that either:

- There is a vertex v such that $x_{t_1} \cap y_{t_2} \cap z_{t_3} = \{v\}$, v is contained in no other edge of T and no vertex different from v is contained in more than one of the three paths.
- P_1, P_2 and $P_3 \setminus \{z_{t_3}\} = (z_2, \dots, z_{t_3})$ are vertex disjoint and $(x_1, \dots, x_{t_1}, z_{t_3}, y_{t_2}, \dots, y_1)$ is a path.

Like in the proof of of Theorem 3 the existence of a double cycle not containing a cycle of independent keys implies the existence of the following structure (see Figure 4):

⁷ Here we use that the length of C_1 is at least 4. If C_1 has length t the fact that x contains three nodes of C_1 only guarantees a cycle of length at most $3 + \lfloor \frac{t-3}{3} \rfloor$.



■ **Figure 4** The case $d = 3$. Non-grey edges: Tridents. Grey edges: Keys that are each dependent on the set of non-black keys.

- **S1:** A trident consisting of three paths $P_1 = (x_1, \dots, x_{t_1})$, $P_2 = (y_1, \dots, y_{t_2})$ and $P_3 = (z_1, \dots, z_{t_3})$ such that the keys of the trident are independent and such that there are, not necessarily distinct, keys x, y, z not in the trident extending the paths P_1, P_2 and P_3 away from their common meeting point such that x, y and z are each dependent on the keys in the trident.

We can bound the probability of this event almost identically to how we proceeded in the proof of Theorem 3. The only difference is that when making the ultimate reduction to the case where $x = y = z = x_1 \oplus y_1 \oplus z_1$ this event is in fact possible (see Figure 4). In this case however, there are three *different* hash function h_x, h_y and h_z such that $h_x(x_1) = h_x(x)$, $h_y(y_1) = h_y(x)$ and $h_z(z_1) = h_z(x)$. However, it is easy to bound the probability that this occur: The number of ways to choose the keys (x, x_1, y_1, z_1) is at most $3^c m^2$ by Lemma 7. The number of ways to choose the hash functions is upper bounded by d^3 . Since the hash functions h_1, \dots, h_d are independent the probability that this can happen in the hash graph is by a union bound at most

$$d^3 3^c m^2 \left(\frac{d}{n}\right)^3 = O(n^{-1})$$

which suffices to complete the proof of Theorem 8.

Summing up

For now we have spent most of our energy proving Theorem 8. At this point it is perhaps not clear to the reader why it is important so let us again highlight the steps to Theorem 1. First of all let $k = \frac{\log \log n}{\log d} + r$ for r a sufficiently large constant. The steps are:

- (1) Show that if some bin has load k then either the hash graph contains a tight subgraph of size $O(k)$ or a certain kind of witness tree T_k .
- (2) Bound the probability that the hash graph contains a T_k by $O((\log \log n)^{-1})$.
- (3) Bound the probability that the hash graph contains a tight subgraph of size $O(k)$ by $O((\log \log n)^{-1})$.

We can now cross (3) of the list. In fact, we have a much stronger bound than we require. The remaining steps as well as the final proofs of Theorem 1 and Theorem 2 are dealt with in the full version of this paper [1]. As already mentioned the proofs of all the above steps (except step (3)) are intricate but straightforward generalisations of the methods in [7].

References

- 1 Anders Aamand, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. *CoRR*, abs/1804.09684, 2018. URL: <https://arxiv.org/abs/1804.09684>.
- 2 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal of Computation*, 29(1):180–200, 1999. See also STOC’94.
- 3 Larry Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. See also STOC’77.
- 4 L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. In *IEEE 52nd Symposium on Foundations of Computer Science*, FOCS, pages 599–608, 2011.
- 5 Xue Chen. Derandomized balanced allocation. *CoRR*, abs/1702.03375, 2017. Preprint. URL: <http://arxiv.org/abs/1702.03375>, arXiv:1702.03375.
- 6 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over k -partitions. In *Proc. 56th Symposium on Foundations of Computer Science*, FOCS, pages 1292–1310, 2015.
- 7 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. In *Proc. 27. ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1631–1642, 2016.
- 8 Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997. doi:10.1006/jagm.1997.0873.
- 9 Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proc. 35th ACM Symposium on Theory of Computing*, STOC, pages 629–638, 2003. doi:10.1145/780542.780634.
- 10 Gaston H. Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM*, 28(2):289–304, 1981.
- 11 Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.
- 12 Michael Mitzenmacher, Andrea W. Richa, and Ramesh Sitaraman. The power of two random choices: A survey of techniques and results. *Handbook of Randomized Computing*, 1:255–312, 2001.
- 13 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.
- 14 Rasmus Pagh and Flemming F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. See also ESA’01.
- 15 Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM*, 59(3):14:1–14:50, 2012. Announced at STOC’11.
- 16 Omer Reingold, Ron D. Rothblum, and Udi Wieder. Pseudorandom graphs in data structures. In *Proc. 41st International Colloquium on Automata, Languages and Programming*, ICALP, pages 943–954, 2014. doi:10.1007/978-3-662-43948-7_78.
- 17 Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal of Computing*, 33(3):505–543, 2004. See also FOCS’89.
- 18 Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. 54th Symposium on Foundations of Computer Science*, FOCS, pages 90–99, 2013.
- 19 Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal of Computing*, 41(2):293–331, apr 2012. Announced at SODA’04 and ALENEX’10.

5:14 Power of d Choices with Simple Tabulation

- 20 Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003. See also FOCS’99.
- 21 Udi Wieder. Hashing, load balancing and multiple choice. *Foundations and Trends in Theoretical Computer Science*, 12(3-4):275–379, 2017. doi:10.1561/04000000070.
- 22 Philipp Woelfel. Asymmetric balanced allocation with simple hash functions. In *Proc. 17th ACM-SIAM Symposium on Discrete Algorithm*, SODA, pages 424–433, 2006.
- 23 Albert L. Zobrist. A new hashing method with application for game playing. Technical report, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.