Hashing, Load Balancing and Multiple Choice DRAFT*

Udi Wieder
VMware Research
udi.wieder@gmail.com

September 19, 2016

Abstract

Many tasks in computer systems could be abstracted as distributing items into buckets, so that the allocation of items across buckets is as balanced as possible, and furthermore, given an item's identifier it is possible to determine quickly into which bucket it was assigned. A canonical example is a dictionary data structure, where 'items' stands for key-value pairs and 'buckets' for memory locations. Another example is a distributed key-value store, where the buckets represent whole servers. A third example may be a distributed execution engine where items represent processes and buckets computational devices, and so on. A common technique in this domain is the use of a *hash-function* that maps an item into a relatively short fixed length string. The hash function is then used in some way to associate the item to its bucket. The use of a hash function is typically the first step in the solution and additional algorithmic ideas are required to deal with collisions and the imbalance of hash values.

In this manuscript we survey some of these techniques. We focus on multiple choice schemes where items are placed into buckets via the use of several independent hash functions, and typically an item is placed at the least loaded bucket at the time of placement. We analyze the distributions obtained in detail, and show how these ideas could be used to design basic data structures.

With respect to data structures we focus on dictionaries, presenting linear probing, cuckoo hashing and many of their variants.

*feedback: errors, typos and omissions, please send to the author

Contents

1 Introduction

'Load Balancing' is a generic name given to a variety of algorithmic problems where a set of items need to be partitioned across buckets, so that the load of each bucket, however defined, is approximately evenly distributed. Phrased in such general terms, the task of load balancing is one of the most fundamental and commonly addressed algorithmic challenges. Typical applications include storage systems where buckets are disks and items files or blocks, data structures where buckets are memory locations and items are keys or distributed execution engines where buckets are servers and items are processes, etc..

This manuscript aims at providing some of the basic algorithmic ideas that underlie many of the practical and theoretically interesting approaches for this problem, focusing on multiple choice schemes and their variants. Our starting point is a balls-into-bins model presented in Section1.1. Throughout Sections 2,3,4 we examine in detail multiple choice techniques for load balancing. We then move to data structures in Section 5, presenting cuckoo-hashing and some of its variants. Finally we discuss the linear probing dictionary, which while not a part of the multiple-choice schema is commonly used and fast in practice.

1.1 The balls-into-bins model

A common framework for reasoning about load balancing processes is that of 'balls' and 'bins' where balls represent the demand (keys, processes, files etc..) and 'bins' represent the supply of resources (table slots, servers, storage units etc..). Throughout this manuscript we use the terms buckets and bins as well as items and balls interchangably.

In this setting we have m balls that are thrown into n bins, typically sequentially according to some allocation rule. The goal is to understand the allocation of balls into bins at the end of the process, usually bounding the load (=number of balls) in the most loaded bin. In this model balls are assigned to bins via one or more *hash functions*. These are functions that map a ball's unique i.d. (typically implicit in the model) to the set of bins. Using a hash function (as opposed to a sample of a bin) is useful in the common case where a ball's location needs to be recovered from its i.d..

The Random Hashing Assumption Throughout most of the manuscript we make the assumption that the hash functions h we use are *completely random*. That is, h(ball.id) is a uniformly sampled bin, independent of $h(\cdot)$

for all other balls. Another way of saying it is that the family of functions H from which h is uniformly sampled is all functions from the universe of bin i.d's to the set of bins. Further, we ignore the time it takes to compute h and the space it takes to store it. This assumption allows us to focus on the probabilistic properties of the allocation while ignoring the details of specifying an explicit function. In practice however a specific and explicit hash function has to be implemented, and one has to take into account not only the probabilistic properties of the hash function but also the space required to store it and the time required to compute it. One can quickly observe that a perfectly random hash function is too expensive to implement in realistic scenarios. A vast body of work is dedicated to removing this assumption and exploring time/space/randomness trade-offs, often for specific applications. The starting point of this line of research is the seminal work of Carter and Wegman [22] on universal hashing. In this manuscript we typically stick with the random hashing assumptions, but for further reading see Section 5.5.

1.2 The Dictionary Data Structure

A *dictionary* is a data structure that stores *key,value* pairs and supports the operations of insertion, deletion and lookup. It is one of the oldest and most widely used data structures already implemented in the 50's c.f [43, 90]. Numerous implementations exist in essentially all standard libraries. There are many possible ways to implement dictionaries with different algorithmic ideas, and we review some of them in Section 5, but as a primer consider the most basic design called a *simple chained hash table*. The design employs a hash function h, that maps the domain of keys to the set $[n]$. An array A of length n is allocated. Ideally we would like a key-value pair (k, v) to be placed in $A[h(k)]$. This is not attainable since more than one pair may be mapped to the same index in the array, a phenomena known as *hash collisions*. In the simple chaining hash table the issue is resolved by letting each element of the array be a head pointer of a linked list which connects all the items mapped to that index of the array.

Under this scheme the running time of the lookup operation is bounded by the number of items mapped to each index of the array. This implementation falls neatly within the balls-into-bins model and is the topic of the next section.

2 Simple Hashing - the One Choice Scheme

In this section we assume each of the m balls is mapped uniformly and independently into one of n bins. The question we ask is: *How many balls will there be in the heaviest loaded bin?* The name *one choice scheme* refers to the fact that only a single bin is sampled per ball. If the sample is done via a hash function this models a simple chained hash table and so the maximal load bounds the worst case running time of the lookup operation.

We denote by $\mathtt{L}_i(m)$ the number of balls mapped to the i'th bin after m balls had been placed. Our object of interest is therefore $\mathtt{L}(m) := \max_i \mathtt{L}_i(m)$. We also denote $\mathtt{Gap}(m) := \mathtt{L}(m) - m/n$.

Let X_i^j be the indicator that the i'th ball was placed in the j'th bin, so $\Pr[X_i^j = 1] = 1/n$ and $\mathtt{L}_j(m) = \sum_{i\in[m]} X_i^j$. We see that the variable $\mathtt{L}_j(m)$ has the binomial distribution $B(m, \frac{1}{n})$, so we can make use of standard bounds on the tails of the Binomial distribution. The following version of Chernoff's theorem is taken from [78].

Theorem 2.1. *Let $X_1, X_2, \ldots, X_m$ be mutually independnet variables such that, for $1 \le i \le m$, $\Pr[X_i = 1] = p_i$ and $\Pr[X_i = 0] = 1 - p_i$, where $0 < p_i < 1$. Then, for $X = \sum X_i$ and $\mu = \mathbb{E}[X] = \sum p_i$ and any $\delta > 0$.*

$$\Pr[X > (1+\delta)\mu] < \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^\mu.$$

We define $\delta(m)$ to be the smallest number such that

$$\Pr[\mathtt{L}_j > \tfrac{m}{n} + \delta(m)] < 1/n^2.$$

By the union bound

$$\Pr\left[\mathtt{L}(m) = \max_j L_j > \frac{m}{n} + \delta(m)\right] < \frac{1}{n}$$

In other words, with probability $\ge 1 - 1/n$, $\delta(m) \ge \mathtt{Gap}(m)$. A simple calculation reveals that $\delta(n) \le O\left(\frac{\ln n}{\ln\ln n}\right)$ and that if $m > n \ln n$ then

$$\delta(m) \le O\left(\sqrt{\frac{m \ln n}{n}}\right)$$

In the following we prove tighter bounds by taking a more careful look at the tail of the Binomial distribution.

Lemma 2.2. *Let $k_\alpha := \alpha \ln n / \ln \ln n$ and let $X = |\{i : \mathtt{L}_i(n) \geq k_\alpha\}|$, i.e., X counts the number of bins with at least k_α balls after n balls had been thrown. Then,*

$$\lim_{n\to\infty} \mathbb{E}[X] = \begin{cases} 0 & \text{if } \alpha > 1 \\ \infty & \text{if } 0 < \alpha < 1 \end{cases}$$

By Markov's inequality $\Pr[X \geq 1] \leq \mathbb{E}[X]$ and since X is a natural number, Lemma 2.2 implies the following upper-bound on the load:

Theorem 2.3. *If $\alpha > 1$*

$$\Pr[\mathtt{L}(n) < k_\alpha] = \Pr[X = 0] \geq 1 - o(1)$$

Proof of Lemma 2.2. Let X_i be the random variable indicating whether $\mathtt{L}_i(n) \geq k_\alpha$ and let $X := \sum X_i$. Clearly,

$$E[X_i] = \Pr[\mathtt{L}_i \geq k_\alpha] = \Pr[B(n, 1/n) \geq k_\alpha].$$

Here we use the notation $B(n,p)$ to denote the Binomial distribution, and with some abuse of notation also a random variable distributed according to it. We need the following lemma.

Lemma 2.4. *For all $h \geq 1$*

$$\Pr\left[B(n, \tfrac{1}{n}) \geq h+1\right] = (1 + O(\tfrac{1}{h})) \Pr\left[B(n, \tfrac{1}{n}) = h+1\right]$$

Proof. For ease of notation define $b(k) := \Pr[B(n, 1/n) = k]$. Observe that for all $k \geq h+1$:

$$\frac{b(k)}{b(k+1)} = \frac{(n-k)(1/n)}{(k+1)(1-1/n)} \leq \frac{1}{k} := \lambda$$

We have:

$$\sum_{k \geq h+1} b(k) \leq b(h+1) \sum \lambda^i$$

Now $\sum \lambda^i = \frac{1}{1-\lambda} = \frac{k}{k-1} = 1 + \frac{1}{k-1} \leq 1 + \frac{1}{h}$ and the claim follows. □

We continue the proof of Lemma 2.2. By Lemma 2.4 we have

$$\begin{aligned} \mathbb{E}[X] &= n \Pr[B(n, \tfrac{1}{n}) \geq k_\alpha] = n(1 + o(1)) b(k_\alpha) \\ &= n(1 + o(1)) \binom{n}{k_\alpha} \left(\frac{1}{n}\right)^{k_\alpha} \left(1 - \frac{1}{n}\right)^{n - k_\alpha} \end{aligned}$$

We recall that $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$ and that $1 + x < e^x$ and compute

$$\begin{aligned} &\leq n(1 + o(1)) \left(\frac{e}{k_\alpha}\right)^{k_\alpha} \\ &\leq \exp(\ln n(1 - \alpha + o(1))) \end{aligned}$$

which has a limit of 0 when $\alpha > 1$. If $\alpha < 1$ a similar calculation, using the fact that $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$ reveals the limit to be ∞. □

Theorem 2.3 is tight, it is possible to show that if $\alpha < 1$, then $\Pr[L(n) < k_\alpha] \leq o(1)$. This does not follow directly from Lemma 2.2, but rather requires an additional bound on the second moment of $L(n)$.

Notes Theorem 2.3 and its analysis are taken from [93]. There they show a slightly tighter bound with $k_\alpha = \frac{\log n}{\log \log n}(1 + \alpha \frac{\log \log n}{\log \log \log n})$. They also show upper and lower bounds for a larger range of m. All the upper bounds use the first moment method as demonstarted above. The second moment method is needed for the matching lower bound. A different way to obtain the lower bound for the $m = n$ case is via Poisson approximation, see for instance [75] §5.4. Poisson approximation is a particularly effective tool when $m < n$.

3 Multiple Choice Schemes

In this section we investigate an interesting allocation process that from an implementational point of view is only a slight modification of the one choice scheme, yet it produces a dramatically more balanced allocation. As before, we assume that the balls are placed sequentially. The twist is that instead of sampling a single bin, whenever a ball u is placed we sample uniformly and independently d bins $b_1(u), \ldots, b_d(u)$. Ball u is placed in the least loaded of these at the time of placement. For concreteness lets assume that ties are broken randomly. This allocation process is known as Greedy[d] or as the *d-choice scheme*. Clearly when $d = 1$ Greedy[d] is simply the 'one-choice scheme' as described before. While from an algorithmic point of view, the change in the allocation process may seem minor, surprisingly, it produces a dramatic improvement in the balance of laod whenever $d > 1$.

From a techincal point view, the proof of Theorem 2.3 relies on the fact that the placement of balls are mutually independent. This does not hold once $d > 1$; the placement of a ball depends not only on its independent

random choices of bins but also on the placement of all balls that were placed before it.

We start with the most basic case where the number of balls m is the same as the number of bins n. This proof was first given in [9] and was also covered in an excellent survey [74]. We provide the basics and move on to newer results.

3.1 The Greedy[d] process

In this section we prove the following:

Theorem 3.1. *After n balls are placed sequentially into n bins using Greedy[d] with $d > 1$, it holds that* $\Pr[\mathtt{L}(n) \geq \frac{\ln \ln n}{\ln d} + O(1)] \leq o(1/n)$

The proof we show is by induction. It is tempting to attempt an induction on n, but that is not the parameter of the induction. Denote by $\nu_i(t)$ the number of bins with load at least i after t balls are thrown. We write ν_i for $\nu_i(n)$. Theorem 3.1 states that with high probability $\nu_k = 0$, for $k = \frac{\ln \ln n}{\ln d} + O(1)$. We will bound ν_i with the induction being on i. Note that $\nu_i(t)$ is non-decreasing in t so it is enough to bound only $\nu_i(n)$, that is, the load at the end of the process. The bounds are probabilist, so one of the technical challenges is to carry the error probability throughout the induction.

Since the total number of balls is n, for every $j > 0$ there could be at most n/j bins with j balls or more. This provides the base case for the induction. Fix some j:

$$\Pr\left[\nu_j \leq \frac{n}{j}\right] = 1 \tag{1}$$

The bound on ν_i is derived in a somewhat roundabout way. Instead of counting bins with load i we count balls which are placed in bins of load at least i. To formalize the idea we need to set some notation. For $t = 1 \ldots n$, the bin in which the t'th ball is placed is denoted by $b(t)$. The *height* of the t'th ball, denoted by $h(t)$, is the number of balls in $b(t)$ at the time of placement plus one. So for instance, the height of the first ball is always 1, and $\mathtt{L}(n) = \max_t h(t)$. Denote by $\mu_i(t)$ the number of balls of height at least i immeadeatly after the t'th ball was placed (we write μ_i for $\mu_i(n)$).Now, every bin of load i must have a ball of height i, therefore:

$$\nu_i(t) \leq \mu_i(t)$$

Let β_i be some upper bound, shown (inductively) on ν_i. Now, consider some ball t. For ball t to be at height $\geq i+1$, it must be that the d bins it sampled all had load $\geq i$. There are at most β_i such bins at time n and therefore at most β_i such bins at any previous time as well. So,

$$\Pr[h(t) \geq i+1 \mid \nu_i \leq \beta_i] \leq \frac{\beta_i^d}{n^d}$$

Notice that μ_{i+1} is the sum of n of these variables so linearity of expectation implies

$$\mathbb{E}[\mu_{i+1} \mid \nu_i \leq \beta_i] \leq \sum \Pr[h(t) \geq i+1 \mid \nu_i \leq \beta_i] \leq \frac{\beta_i^d}{n^{d-1}} \tag{2}$$

Now we can state explicitly the values of β_i. As implied by (1), we set $\beta_4 = n/4$, and we set $\beta_{i+1} = \frac{2\beta_i^d}{n^{d-1}}$. Note that $\mathbb{E}[\mu_{i+1}|\nu_i \leq \beta_i] \leq \beta_{i+1}/2$. We want to show that ν_i is bounded by β_i with high probablity. The analysis is split to two. First, as long as β_i is not too small, concentration bounds drive the induction. When β_i is small a more brute force approach suffices.

The following Lemma establishes the breaking point between the two cases.

Lemma 3.2. *Let i^* be the largest value of i such that $\beta_i \geq 6 \ln n$. Then, $i^* = \ln\ln n / \ln d + O(1)$.*

Proof. We show by induction that

$$\beta_{i+4} = \frac{n}{2^{2d^i - \sum_{j=0}^{i-1} d^j}}$$

We have defined $\beta_4 = n/4$, so this holds for $i = 0$. By definition $\beta_{(i+1)+4} = \frac{2\beta_{i+4}^d}{n^{d-1}}$ which by the inductive hypothesis equals

$$= 2\left(\frac{n}{2^{2d^i - \sum_{j=0}^{i-1} d^j}}\right)^d / n^{d-1} = \frac{n}{2^{2d^{i+1} - \sum_{j=0}^{i} d^j}}$$

This implies $\beta_i \leq n/2^{d^{i-4}}$ and the claim follows. □

Lemma 3.3. *For $i < i^*$, $\Pr[\nu_i \geq \beta_i] \leq \frac{i}{n^2}$.*

Proof. Since the random choices of balls are independent of one another, it follows that

$$\Pr[\mu_{i+1} > \beta_{i+1} \mid \nu_i \leq \beta_i] \leq \Pr[B(n, \frac{\beta_i^d}{n^d}) > \beta_{i+1}]$$

Chernoff Bound implies this is smaller than $1/n^2$ whenever $i \leq i^*$. Now we have by induction:

$$\begin{aligned}\Pr[\nu_{i+1} \geq \beta_{i+1}] &\leq \Pr[\mu_{i+1} \geq \beta_{i+1}] \\ &\leq \Pr[\mu_{i+1} > \beta_{i+1} \mid \nu_i \leq \beta_i] + \Pr[\nu_i > \beta_i] \\ &\leq \frac{1}{n^2} + \frac{i}{n^2} = \frac{i+1}{n^2}\end{aligned}$$

□

All that is required to complete the proof of Theorem 3.1 is to handle the case $i > i^*$. We have already established that w.h.p $\nu_{i^*} \leq 6 \log n$. In that case, the probability a bin with such a load is assigned an additional ball is at most $\left(\frac{6 \log n}{n}\right)^2$. Since the total number of balls is n, the expected total number of additional balls assigned to such a bin is bounded by $\frac{(6 \log n)^2}{n}$ and is $O(1)$ with probability $o(1/n)$.

We note that the same proof could be extended to the case $m = cn$ for some $c \geq 1$. The only difference is in Equation(1) which is now

$$\Pr\left[\nu_{j+c} \leq \frac{cn}{j+c}\right] = 1$$

and Theorem 3.1 states that $\Pr[\mathtt{L}(cn) \geq \frac{\ln \ln n}{\ln d} + c + O(1)] \leq o(1/n)$. This bounds becomes weaker as c gets larger. We will see in Section 4.2 a better way of dealing with that case.

A lower bound Theorem 3.1 is tight in that with high probability the maximal load is at least $\log \log n / \log d - O(1)$. A more general statement is presented in Section 4.2.3. The general idea is to prove a lower bound inductively in a way much similar to the way the upper bound was proven above.

3.2 The Left[d] Process

Stepping back, the layered induction approach relies on two statements. The first is a base case for the induction, as manifested in (1). Note that Equation (1) is robust, it only relies on the assumption that $m = n$ and does not depend on the specific allocation rule. The second requirement is some induction step as expressed in Equation (2). It turns out that different processes may imply other induction steps that lead to different

bounds. Vöcking [103] suggested a slightly different process named Left[d] and surprisingly showed it has better bounds. The analysis we show is taken from [77].

In this model the n bins are partitioned into d sets $S_1, ..., S_d$, each of size n/d. We assume for simplicity that d divides n. The allocation process places the balls sequentially one by one. Each time step it samples d bins, one bin uniformly and independently from each set. The ball is placed in the least loaded of the d bins sampled. The interesting twist is in the way ties are broken. If multiple bins have minimal load the ball is placed in the bin which belongs to the set with minimal index. It is convenient to think of the sets as placed linearly, so S_1 is the leftmost, to its right is S_2 and so on. Now the tie breaking rule could be phrased as placing the ball in the bin which is leftmost of of the tied bins; hence the name of the scheme.

Theorem 3.4. *After n balls are placed sequentially into n bins using the d-choice scheme with $d > 1$, it holds that* $\Pr[\mathcal{L}(n) \geq \frac{\log\log n}{d \ln \phi_d} + O(1)] \leq o(1/n)$ *where ϕ_d is some constant that depends only on d and is always in* $(1.6, 2)$.

Define X_{jd+k} to be the number of bins from S_k with load at least j, and set $x_i = X_i/n$. For a ball to be at height j in set k it has to sample a bin of height at most $j-1$ from $S_1, \ldots, S_{k-1}$ and a bin of height at most j from sets $S_k, S_{k+1}, \ldots, S_d$. A moment's thought reveals the following recursive relation which is the analog of Equation (2):

$$\mathbb{E}[x_i \mid x_{<i}] \leq d^d \prod_{j=i-d}^{i-1} x_j \tag{3}$$

As before this suggests choosing the β_i that bound x_i such that $\beta_i = 2d^d \prod_{j=i-d}^{i-1} \beta_j$. Simlilary we need to set the base case. As is the case for Equation (1), since there are in total n balls, there are at most $n/4$ bins of load at most $n/4$, so with probability 1,

$$\sum_{i \in [d]} x_{4d+i} \leq \frac{1}{4} \tag{4}$$

Now consider the bound implied by (3) on the expected value of x_{5d+i}. Equation (4) implies it is maximized when the weight of each $x_{4d+i} = 1/4d$. So, for each $i \in [d]$

$$\mathbb{E}[x_{5d+i}] \leq d^d \left(\frac{1}{4d}\right)^d \leq \left(\frac{1}{4}\right)^d$$

So naturaly, for $i \in [d]$ we set $\beta_{5d+i} = 2\left(\frac{1}{4}\right)^d$.

From this point on the proof continues exactly as before, with Chernoff bound implying the accumulated error is small, as long as $\beta_i \geq \frac{6 \log n}{n}$. The bound for the case $\beta_i < \frac{6 \log n}{n}$ is identical to that of Theorem 3.1.

What remains is to understand how fast do the β_i's decrease. Surprisingly, the rucursive equation of (3) decreases faster than (2). The generalized Fibonacci number $F_d(k)$ is defined by setting $F_d(k) = 0$ for $k \leq 0$, and $F_d(1) = 1$. For $k > 1$, $F_d(k) = \sum_{i=1}^{d} F_d(k-i)$. A simple induction implies that:

Lemma 3.5.

$$\beta_i \leq (4d)^{-F_d(i-6d)+1}$$

Let ν_i denote the fraction of bins with load at least i in the system. Then, with high probability:

$$\nu_i = \sum_{j=1}^{d} x_{di+j} \leq d(4d)^{-F_d(di-6d)+1}$$

so the decrease is exponential in $F_d(d \cdot i)$. Define $\phi_d = \lim_{k\to\infty} \sqrt[k]{F_d(k)}$. It is well known that for every $d \geq 2$ the limit ϕ_d exists and is in $(1.6, 2)$, c.f. [40]. This means that the i^* for which we need to switch the analysis to a union bound is at $\frac{\ln \ln n}{d \ln \phi_d} + O(1)$ as is stipulated by Theorem 3.4.

3.2.1 A better dictionary

Multiple choice schemes naturally extend the idea of a simple chained hash table as described in section 1.2. Using multiple hash functions would shorten the length of the chains and thus reduce the worse case lookup time. This approach is often called *d-way chaining*. Alternately, a fixed number of memory slots could be allocated in each bucket and using multiple hash function would decrease the number of items that can not be inserted becasue all slots are full. In this case the difference between Left[d] and Greedy[d] could be significant. See for instance [19].

3.3 Notes and Other Generalizations

A proof of Theorem 3.1 for the case $d = 2$ first appeared in a slightly different form in [57]. The general version and the proof we bring here are from [9]. The Left[d] scheme was shown in [103], with a different proof technique. The proof we show was taken from [77].

3.3.1 Alternative proof techiques

The inductive proof technique which we used above is not the only way to approach Greedy[d] and its variants. A powerful alternative is to explicitly track the combinatoric structures which underlie the process. Here is a rough sketch of the approach. Consider a labeled graph with n nodes and m edges. The nodes represent the n bins. The m edges represent the balls; an edge representing ball u connects the two nodes which represent u's two random bin choices in Greedy[2]. The main idea is to show connections between the load obtained by Greedy[2] (or some vairant of it) and combinatorial properties of this random graph. Consider for instance a ball u and its connected component in the graph C_u. If u is of height ℓ its two random bins were of height at least $\ell - 1$ when u was placed. Track the edges corresponding to the two balls at height $\ell - 1$ in these bins and continue. All the edges discovered this way belong to C_u and the conclusion is that either C_u is large (containing roughly 2^ℓ nodes) or is dense with many cycles. A direct combinatorial proof aims at showing that this is unlikely for large enough ℓ and thus a bound on ℓ is obtained. The sketch above is quite general and could be used in many ways. In fact, the same general approach is used to prove bounds on cuckoo hashing, which we will see next section. There the graph is called the 'cuckoo graph'. There we will see that if $m < n/3$ it is unlikely that C_u is large, and this already provides quick and loose bound of $O(\log \log n)$. The advantage of this approach is that tracking the combinatorial structures explicitly is useful when considering explicit hash functions. See for instance [94] and [27]. The same high-level approach but with a more careful analysis is often referred as *the witness tree* approach and can yield tight results. The proof of Theorem 3.4 in [103] uses this approach. It is especially useful when considering dynamic models which include ball removals. Examples and references could be found in the survey [74].

A different proof technique uses differential equations and fluid limit theorems. The idea is to view the system as growing to infinity and thus the affect of each ball placement becomes infinitesimal and the process becomes continuous. The continuous process could be described using differential equations and analytically solved. Typically the differential equations resemble the recursive relations used in the inductive proof technique which was used here. The main advantages of this techinique are its adaptability to generalizations especially around queuing theory, and its accuracy in providing estimates of the fraction of bins with intermediate load levels. Again, see the survey [74] for many references and examples.

3.3.2 Related processes

Many variations of the basic process Greedy[d] were suggested, typically with the goal of capturing a more realistic model of some algorithm or system. We sketch a few.

In [10] 'chains' of length ℓ are placed. That is, each placement operation entails placing ℓ balls across ℓ consecutive bins. There are $d \geq 2$ potential places for the head of the chain. The chain is placed in the location that minimized the load of the most loaded bin across the ℓ bins that hold the chain. It is shown that if m chains of length ℓ are placed and $m \cdot \ell = O(n)$ then with high probability the maximum load is $(\ln \ln m)/\ln d + O(1)$.

In [58] they consider the following variant of Left[d] . Bins are partitioned to $2n/d$ groups of $d/2$ bins. A ball is placed by sampling two groups, and assigning the ball to the least loaded bin in the lesser loaded group (breaking ties to the left). Surprisingly the maximal load remains similar to that of Theorem 3.4. Both schemes probe the load of d bins, but while in Left[d] the d bins are random, the partition into groups is implies that only 2 memory probes are random while the rest are likely to reside in teh cache, and therefore it runs faster in practice.

In [11] bins have different capacities. Capacity affects the probability a bin is sampled with the probability being proportional to the capacity. The load of a bin is calculated to be the number of balls assigned to it divided by its capacity. It is shown that if the number of balls is equal the total capacity (a requirement analogous to $m = n$) then the maximal load is $\log \log n + O(1)$.

In [73] the d choices of Greedy[d] are not sampled independently, but rather, only two hash functions h_1, h_2 are used. For each ball u two d bins used are $(h_1(u)+ih_2(u)) \mod n$ for $i = 0, 1, \ldots, d-1$. Surprisingly, it shown that the allocation obtained is very similar to that of standard Greedy[d] , not only in terms of the maximal load but also for intermediate values as well.

In [15] [17] they propose a local search algorithm for placement. In this scheme the bins are associated with the nodes of a graph. A ball is hashed to a random bin (node) and then starts a local search around the node until it finds a local minima where it is permanently placed. The maximal load is $O(\log \log n)$ if the graph is constant degree expander.

A separate line of research deals with the case where the balls are not placed sequentially but rather in parallel. Here every round the yet unplaced balls interact with a small number bins and after each communication round a some balls are placed in the bins. The object is to minimize the number

of rounds of communication and total number of messages sent. In [68] it was shown that when $m = n$ a load of 2 could be achieved in $\log^* n + O(1)$ communication rounds and $O(n)$ messages. They also show the bound is tight.

4 The Heavily Loaded case

The layered induction proof techniqe which was presented in Section 3.1 is inherently limited to the case $m = n$, and could easily be extended to $m = O(n)$. The case $m >> n$ is important and captures typical load balancing challenges in large sytstems where multiple items are placed per host. For instance, in a large distributed block storage system, if balls represents blocks and bins represent hosts, it is reasonable to assume that there are about a million balls per bin. We refer to this setting as the 'heavily loaded' case. Many problems arise when trying to extend the previous inductive proof to the heavily loaded case. For instance, how can we carry the error of Lemma 3.3 across an arbitrarily large number of balls. Even more basically, both the inductive base case (1) and the inductive step (2) rely on the number of balls being linear in the number of bins. A breakthrough was achieved by Berenbrink *et al.* in [12]. Recall that $L(m)$ measures the maximal number of balls assigned to any bin after m balls had been placed, and that $\texttt{Gap}(m) := L(m) - m/n$ measures the difference between the maximum and the average load.

Theorem 4.1. *For every $c > 0$ there is a $\gamma = \gamma(c)$ so that for any natural m, after m balls are placed with Greedy[d]*

$$\Pr\left[\texttt{Gap}(m) \geq \frac{\log\log n}{\log d} + \gamma\right] \leq n^{-c}$$

Contrast this with the one choice case in which $\texttt{Gap}(m)$ diverges with the number of balls.

At a high level the approach taken in [12] is the following: first it is shown that for any m, the distribution of $\texttt{Gap}(m)$ is close to the distribution of $\texttt{Gap}(m')$ where $m' = poly(n)$. This is essentially a bound on the mixing time of the underlying Markov chain. The second step is to extend the inductive technique of Theorem 3.1 to the case of $m = poly(n)$. This turns out to be a major technical challenge which manipulates four inductive invariants and uses some computer aided calcualtions. In this survey we follow a much simpler approach presented in [100], and show a slighlty weaker result:

Theorem 4.2. *For any m, after m balls are placed using Greedy[d] it holds that*

$$\mathbb{E}[\texttt{Gap}(m)] \leq \frac{\log\log n}{\log d} + O(\log\log\log n)$$

In the following section we build a machinary that handles the heavily loaded case in a much more general setting, finally proving Theorem 4.2 in Section 4.2. The mechanism we show is taken largely from [89].

4.1 General Placement Processes

An important property of Greedy[d] is that the probability that a given bin gets a ball depends only upon the fraction of bins which have a higher load. Another way to characterize Greedy[d] is via a probability vector $\mathbf{p} = (p_1, \ldots, p_n)$: we order the bins from the most loaded to the least loaded (ties are broken according to some fixed ordering of the bins) and set p_1 to be the probability that the most loaded bin receives the ball, p_2 is the probability that the second bin receives it, and so on. It is easy to see that in Greedy[d] $p_i = \left(\frac{i}{n}\right)^d - \left(\frac{i-1}{n}\right)^d$. So for $d = 1$, $p_i = \frac{1}{n}$ for all i, while if $d > 1$ then $p_i > p_j$ for $i > j$. In other words, when $d > 1$ the process has a *bias towards the light bins*. At a high level, this bias explains the stark difference between the cases $d = 1$ and $d > 1$.

Our approach is to view the process as an n-dimensional version of a biased random walk on a line: Consider a random walk on the line which starts at 0 and each step moves either right or left with equal probability. It is well known that after m steps, the expected distance from 0 is roughly $\sqrt{m}$. If however the random walk has a *bias towards* 0 then the expected distance from 0 is bounded by some constant independent of m. When $n = 2$, the difference between the loads of the two bins is indeed a random walk on the line: an unbiased one in Greedy[1] and a biased one in Greedy[d] for $d > 1$. We reduce the process at hand to a one dimensional process via a potential function $\Gamma : \mathbb{R}^n \to \mathbb{R}$ from the set of allocations to the real numbers such that (i) if Γ is small then the allocation is balanced, and (ii) the expectation of Γ is bounded by a constant independent of m. The main advantage of this approach is that it generalizes to more vectors $\mathbf{p}$.

Weighted balls. In practice it is often the case that items have heterogeneous sizes. Consider for instance the case of a storage system which places files (balls) into servers (bins). In this case each file has a different size, and the load of the server is the total size of files assigned to it. If each ball is assigned an arbitrary, possibly advarsarial, then no non-trivial bounds are

known. Further, some interesting and pathological scenarios are demonstrated in [13]. In this survey we examine a strochastic model instead. We assign each ball a weight by independently sampling from a weight distribution $\mathcal{D}$ and define the load of a bin to be the sum of weights of balls assigned to it. Talwar and Wieder [99] show that in the two choice scheme, if the weight distribution has a finite variance and is 'smooth' in some mild sense, then $\texttt{Gap}(m)$ is independent of m, the number of balls thrown. However, no non-trivial upper bounds on the gap for specific distributions were shown there.

Definition 1. *A* general placement process *is characterized by a probability vector* $\mathbf{p}$ *and a weight distribution* $\mathcal{D}$ *over* $\mathbb{R}_{>0}$*. m balls are placed sequentially into n bins, where for each ball v:*

1. *Bins are sorted by load, where a bin's load is the sum of weights of balls asigned to it. An index $i \in [n]$ is sampled according to* $\mathbf{p}$*, that is, i is sampled with probability p_i*
2. *A weight W is sampled from $\mathcal{D}$ and is asigned to v*
3. *Ball v is placed in the i'th loaded bin*

We make the following assumptions on $\mathbf{p}$

$$p_i \leq p_{i+1} \text{ for } i \in [n-1] \tag{5}$$

For some $\frac{1}{4} > \epsilon > 0$ it holds that

$$p_{\frac{n}{3}} \leq \frac{1-4\epsilon}{n} \text{ and } p_{\frac{2n}{3}} \geq \frac{1+4\epsilon}{n} \tag{6}$$

The first assumption says that the probability a bin receives a ball is non increasing with the bin's load. The second assumption states that the allocation rule strictly prefers the least loaded third of bins over the most loaded third. Note also that these assumptions imply that $\sum_{i \geq \frac{3n}{4}} p_i \geq \frac{1}{4} + \epsilon$ and $\sum_{i \leq \frac{n}{4}} p_i \leq \frac{1}{4} - \epsilon$, a fact that turns out to be useful.

For the distribution $\mathcal{D}$, we first assume that $\mathbb{E}[W] = 1$, where W is drawn from $\mathcal{D}$. This assumption is without loss of generality, because all weights could be scaled appropriately and $\texttt{Gap}$ is then scaled by the same factor. The second assumption we make is that there is a $\lambda > 0$ such

that the moment generating function $M(\lambda) = \mathbb{E}[e^{\lambda W}] < \infty$. Essentially this assumption means that the tail of the distribution decreases at a rate which is at least exponential. In particular, the Exponential, Geometric and Normal distributions have this property. Note that

$$M''(z) = \mathbb{E}[W^2 e^{zW}] \leq \sqrt{\mathbb{E}[W^4]\mathbb{E}[e^{2zW}]}.$$

The above assumption implies that there is an $S \geq 1$ such that for every $|z| < \lambda/2$ it holds that $M''(z) < 2S$. Let $\alpha = \min(\frac{\epsilon}{6S}, \lambda/2)$. Ultimately, α and S are constants that depend only on $\mathcal{D}$ and ϵ. If we have uniform weights, i.e., $\mathcal{D}$ is concentrated on 1, then we can take $S = 1$, $\alpha = \epsilon/6$. Let $\texttt{Gap}'(m)$ denote the difference between the maximal load and the minimal load. The remainder of this section is dedicated to proving the following theorem:

Theorem 4.3. *Under the assumptions above, for every m,*

$$\mathbb{E}[\textit{Gap}'(m)] \leq \tfrac{\log n}{\alpha} + O(\tfrac{\log(1/\alpha\epsilon)}{\alpha})$$

Moreover,

$$\Pr\left[\textit{Gap}'(m) > \tfrac{2\log n}{\alpha} + O(\tfrac{\log(1/\alpha\epsilon)}{\alpha})\right] \leq \tfrac{1}{n}$$

Let us check the bounds the above results give for some simple distributions.

Example 2. *Let $\mathcal{D}$ be the distribution that is zero with probability $(1 - \frac{1}{K})$ and K with probability $\frac{1}{K}$. For this distribution, we can check that $M''(\frac{1}{K})$ is in $O(K)$, so that we can set $\alpha = \Theta(\frac{\epsilon}{K})$. This leads to a gap of $O(K \log n/\epsilon)$, which is tight up to constants.*

Example 3. *Let $\mathcal{D}$ be the exponential distribution with parameter 1. Then $M''(1/2)$ is finite. Thus the gap is $O(\log n/\epsilon)$. This is tight up to constants.*

Moreover, note that the distribution may be different in each time step, as long as the independence is preserved and the bound on the moment generating function holds.

Recal, that our approach is to reduce the process at hand to a one dimensional process via a potential function $\Gamma : \mathbb{R}^n \to \mathbb{R}$ from the set of allocations to the real numbers such that (i) if Γ is small then the allocation is balanced, and (ii) the expectation of Γ is bounded by a constant independent of m. We do that next.

The Underlying Markov Chain: We mark time t, where each time unit one ball is placed, so at time t, t balls have been placed. Let $x(t)$ be the vector denoting the load of each bin minus the average load, at time t. So $\sum x_i(t) = 0$. Under this notation, a bin that trails behind the average has negative load. We assume that x is sorted so that $x_i \geq x_{i+1}$ for $i \in [n-1]$. The process defines a Markov chain over the vectors $x(t)$ as follows:

- sample $j \in_{\mathbf{p}} [n]$, i.e. pick j with probability p_j.
- sample $W \in \mathcal{D}$
- set $y_i = x_i(t) + W - \frac{W}{n}$ for $i = j$ and $y_i = x_i(t) - \frac{W}{n}$ for $i \neq j$
- obtain $x(t+1)$ by sorting y

Define the following potential functions

$$\Phi(t) = \Phi(x(t)) := \sum_{i=1}^{n} \exp(\alpha x_i)$$
$$\Psi(t) = \Psi(x(t)) := \sum_{i=1}^{n} \exp(-\alpha x_i)$$
$$\Gamma(t) = \Gamma(x(t)) := \Phi(x(t)) + \Psi(x(t))$$

Note that $\Gamma(0) = 2n$. We show that if $\Gamma(x(t)) \geq cn$ for some $c > 0$ then $\mathbb{E}\left[\Gamma(x(t+1)) \mid x(t)\right] \leq \Gamma(x(t))(1 - \frac{\alpha\epsilon}{4n})$. We will use this to show that for every given t, $\mathbb{E}[\Gamma(t)]$ is bounded.

One may think that Φ would make a more natural choice. Φ however is barely affected by the load of the bins that are dragged behind the average (and thus have negative load in the load vector). This results in pathological vectors x for which Φ can increase in expectation even when it is arbitrarily large. Another natural choice of a potential funciton is $\sum \exp(|\alpha x_i|)$. This function is similar to Γ. We chose Γ because it allows us to investigate Φ and Ψ separately.

We start by calculating the expected change of Φ and Ψ individually. We write Φ instead of $\Phi(t)$ when t is clear from context.

Lemma 4.4. *For Φ defined as above,*

$$\mathbb{E}[\Phi(t+1) - \Phi(t) \mid x(t)] \leq \sum_{i=1}^{n} \left(p_i(\alpha + S\alpha^2) - (\tfrac{\alpha}{n} - S\tfrac{\alpha^2}{n^2})\right) e^{\alpha x_i}. \qquad (7)$$

Proof. Let Δ_i denote the change in $\Phi_i = \exp(\alpha x_i)$, i.e. $\Delta_i = \exp(\alpha y_i) - \exp(\alpha x_i)$, where $y_i = x_i + W - \frac{W}{n}$ with probability p_i, and $y_i = x_i - \frac{W}{n}$ otherwise. Recall that $M(z) := \mathbb{E}[\exp(zW)]$. In the first case, when the ball is placed in bin i, we use Maclaurin expansion and calculate the expected change (taken over randomness in W) Δ_i:

$$\begin{aligned}\mathbb{E}[e^{\alpha(x_i+W-\frac{W}{n})}] - e^{\alpha x_i} &= e^{\alpha x_i}(M(\alpha(1-\tfrac{1}{n})) - 1) \\ &= e^{\alpha x_i}(M(0) + M'(0)\alpha(1-\tfrac{1}{n}) + M''(\zeta)(\alpha(1-\tfrac{1}{n}))^2/2 - 1)\end{aligned}$$

for some $\zeta \in [0, \alpha(1-\frac{1}{n})]$. By the assumption on $\mathcal{D}$ and α, $M''(\zeta) \leq 2S$. Moreover, $M(0) = 1$ and $M'(0) = \mathbb{E}[W] = 1$. Thus the above expression can be bounded from above by

$$e^{\alpha x_i}(\alpha(1-\tfrac{1}{n}) + S\alpha^2) \tag{8}$$

Similarly, in the case that the ball goes to a bin other than i, the expected value of Δ_i is bounded by

$$(-\tfrac{\alpha}{n} + S\tfrac{\alpha^2}{n^2})e^{\alpha x_i} \tag{9}$$

Adding (8) and (9)

$$\mathbb{E}[\Delta_i] \leq p_i(\alpha(1-\tfrac{1}{n}) + S\alpha^2)e^{\alpha x_i} - (1-p_i)(\tfrac{\alpha}{n} - S\tfrac{\alpha^2}{n^2})e^{\alpha x_i} \leq \left(p_i(\alpha + S\alpha^2) - (\tfrac{\alpha}{n} - S\tfrac{\alpha^2}{n^2})\right)e^{\alpha x_i}.$$

The claim follows. □

We can now bound the expected increase Φ.

Corollary 4.5.

$$\mathbb{E}[\Phi(t+1) - \Phi(t) \mid x(t)] \leq \frac{2\alpha}{n}\Phi(t) \tag{10}$$

Proof. Note that $S\alpha < 1$ so (7) implies

$$\mathbb{E}[\Phi(t+1) - \Phi(t) \mid x(t)] \leq \sum_{i=1}^{n} 2\alpha p_i e^{\alpha x_i}.$$

For every $i \in [n-1]$ we know that $p_i \leq p_{i+1}$. Let $p' = \frac{p_i + p_{i+1}}{2}$. Since $x_i \geq x_{i+1}$ we have

$$p'e^{\alpha x_i} + p'e^{\alpha x_{i+1}} \geq p_i e^{\alpha x_i} + p_{i+1}e^{\alpha x_{i+1}}$$

In other words, for given decreasing x_is, and under the constraints that the p_i's are increasing the expression above is maximized when $p_i = 1/n$ for all i. The claim follows. □

Similar arguments show that

Lemma 4.6. *Let Ψ be defined as above. Then*

$$\mathbb{E}[\Psi(t+1) - \Psi(t) \mid x(t)] \leq \sum_{i=1}^{n} \left(p_i(-\alpha + S\alpha^2) + (\tfrac{\alpha}{n} + S\tfrac{\alpha^2}{n^2}) \right) e^{-\alpha x_i}. \quad (11)$$

Corollary 4.7.

$$\mathbb{E}[\Psi(t+1) - \Psi(t) \mid x(t)] \leq \frac{2\alpha}{n}\Psi(t) \quad (12)$$

Proof. This follows immediately as $p_i > 0$ and $S\alpha < \frac{1}{6}$. □

So far we showed that Φ and Ψ don't increase by much on expectation. Ultimately our goal is to show that if large they are actually expected to *decrease*. We start by showing this under the assumption that the allocation itself $x(t)$ is reasonable balanced. More precisely, if $x_{\frac{3n}{4}} \leq 0$ (so the least loaded quorter of bins all have load at most average), then Φ decreases in expectation, and if $x_{\frac{n}{4}} \geq 0$, (so the most loaded quorter of bins have load at least average), then Ψ decreases in expectation.

Lemma 4.8. *Let Φ be defined as above. If $x_{\frac{3n}{4}}(t) \leq 0$, then $\mathbb{E}[\Phi(t+1) \mid x(t)] \leq (1 - \frac{\alpha\epsilon}{n})\Phi(t) + 1$.*

Proof. We bound the size of each of the terms in (7) separately. First we bound from above $\sum_{i=1}^{n} p_i(\alpha + S\alpha^2)e^{\alpha x_i}$ for a fixed $\Phi(x)$, for x which is non increasing with $\sum_i x_i = 0$.

$$\begin{aligned} \sum_{i=1}^{n} p_i(\alpha + S\alpha^2)e^{\alpha x_i} &\leq \sum_{i<\frac{3n}{4}} p_i(\alpha + S\alpha^2)e^{\alpha x_i} + \sum_{i\geq\frac{3n}{4}} p_i(\alpha + S\alpha^2)e^{0} \\ &\leq \sum_{i<\frac{3n}{4}} p_i(\alpha + S\alpha^2)e^{\alpha x_i} + 1 \quad (13) \end{aligned}$$

since $\alpha + S\alpha^2 \leq \frac{6\epsilon+\epsilon^2}{36S} \leq 1$ by our assumptions that $\epsilon \leq 1$ and $S \geq 1$.

Now set $y_i := e^{\alpha x_i}$. The first term above is no larger than the maximum

value of

$$(\alpha + S\alpha^2) \sum_{i<\frac{3n}{4}} p_i y_i$$

subject to

$$\sum_{i<\frac{3n}{4}} y_i \leq \Phi$$

$$y_{i-1} \geq y_i \quad \forall\ 1 < i < \tfrac{3n}{4}.$$

Since $\mathbf{p}$ is non-decreasing and $\mathbf{y}$ is non-increasing, the maximum is achieved when $y_i = \frac{4\Phi}{3n}$ for each i, and is at most $(\alpha + S\alpha^2)(\frac{3}{4} - \epsilon)\frac{4\Phi}{3n}$.

We can now plug this bound in (13), and substituting in (7), the expected change in Φ is bounded by

$$\begin{aligned}
\mathbb{E}\left[\Phi(t+1) - \Phi(t) \mid x(t)\right] &\leq (\alpha + S\alpha^2)(\frac{3}{4} - \epsilon)\tfrac{4\Phi}{3n} - \left(\frac{\alpha}{n} - S\frac{\alpha^2}{n^2}\right)\Phi + 1 \\
&\leq \frac{\alpha\Phi}{n}\big((1 + S\alpha)(1 - \frac{4\epsilon}{3}) - 1 + S\frac{\alpha}{n}\big) + 1
\end{aligned}$$

Assuming $S\alpha \leq \epsilon/6$ we have

$$\begin{aligned}
&\leq \frac{\alpha}{n}\Phi\left(\tfrac{\epsilon}{6} - \tfrac{4\epsilon}{3} + \tfrac{\epsilon}{6n}\right) + 1 \\
&\leq -\frac{\alpha\epsilon}{n}\Phi + 1
\end{aligned}$$

The claim follows. □

A similar bound is shown for Ψ:

Lemma 4.9. *Let Ψ be defined as above. If $x_{\frac{n}{4}}(t) \geq 0$, then $\mathbb{E}[\Psi(t+1) \mid x(t)] \leq (1 - \frac{\alpha\epsilon}{n})\Psi(t) + 1$.*

Proof. We first show an upper bound for $\sum_{i=1}^{n} p_i(-\alpha + S\alpha^2)e^{-\alpha x_i}$ for a fixed $\Psi(x)$. Recall that x is non increasing and $\sum_i x_i = 0$. Since $(-\alpha + S\alpha^2)$ is negative, we have

$$\sum_{i=1}^{n} p_i(-\alpha + S\alpha^2)e^{-\alpha x_i} \leq (-\alpha + S\alpha^2)\sum_{i\geq\frac{n}{4}} p_i e^{-\alpha x_i}$$

Now set $z_i := e^{-\alpha x_i}$. Under the assumption on $x_{\frac{n}{4}}$, the sum $\sum_{i\geq\frac{n}{4}} z_i$ is at least $\Psi - \frac{n}{4}$. Since $(-\alpha + S\alpha^2)$ is negative, to bound the second term, we

need to find the minimum value of

$$\sum_{i \geq \frac{n}{4}} p_i z_i$$

subject to

$$\sum_{i \geq \frac{n}{4}} z_i \geq \Psi - \frac{n}{4}$$

$$z_{i-1} \geq z_i \quad \forall\ i > \tfrac{n}{4}.$$

Since both $\mathbf{p}$ and $\mathbf{z}$ are (weakly) increasing, the minimum is achieved when $z_i = \frac{4(\Psi - \frac{n}{4})}{3n}$ for each i. Using the assumption that $\sum_{i \geq n/4} p_i \geq \frac{3}{4} + \epsilon$ we can bound the expression above by $(-\alpha + S\alpha^2)(\frac{3}{4} + \epsilon)\frac{4(\Psi - \frac{n}{4})}{3n}$. We can now bound the expected change in Ψ by plugging this bound in (11).

$$\begin{aligned}
\mathbb{E}[\Psi(t+1) - \Psi(t) \mid x(t)] &\leq (-\alpha + S\alpha^2)(\tfrac{3}{4} + \epsilon)\tfrac{4(\Psi - \frac{n}{4})}{3n} + \tfrac{\alpha}{n}(1 + S\tfrac{\alpha}{n})\Psi \\
&= \tfrac{\alpha}{n}\left((1 + S\tfrac{\alpha}{n})\Psi + (-1 + S\alpha)(\tfrac{3}{4} + \epsilon)\tfrac{4\Psi - n}{3}\right) \\
&= \tfrac{\alpha}{n}\left((1 + S\tfrac{\alpha}{n})\Psi + S\alpha(\tfrac{3}{4} + \epsilon)\tfrac{4\Psi - n}{3} - (\tfrac{3}{4} + \epsilon)\tfrac{4\Psi - n}{3}\right) \\
&\leq \tfrac{\alpha\Psi}{n}\left(1 + S\tfrac{\alpha}{n} + S\alpha(\tfrac{3}{4} + \epsilon)\tfrac{4}{3} - (\tfrac{3}{4} + \epsilon)\tfrac{4}{3}\right) + \frac{\alpha}{3}(\tfrac{3}{4} + \epsilon) \\
&\leq -\frac{\alpha\epsilon}{n}\Psi + 1
\end{aligned}$$

where the last inequality follows since $\epsilon \leq \frac{1}{4}$ and $S\alpha \leq \frac{\epsilon}{6}$.

□

So far we have shown that if the allocation is more or less balanced, both Φ and Ψ decrease on expectation. When the allocation is extremely unbalanced one of these potential function may increase while the other decreases. The idea of the proof is to show that the decreasing function dominates the increasing one. The next lemma will be useful in the case that $x_{\frac{3n}{4}} > 0$. For ease of notation we write $\Delta\Phi$ to denote $\Phi(x(t+1)) - \Phi(x(t))$.

Lemma 4.10. *Suppose that $x_{\frac{3n}{4}} > 0$ and $\mathbb{E}[\Delta\Phi | x(t)] \geq -\frac{\alpha\epsilon}{n}\Phi$. Then either $\Phi < \frac{\epsilon}{4}\Psi$ or $\Gamma < cn$ for some $c = poly(\frac{1}{\epsilon})$.*

The lemma shows that if Φ does not decrease enough by expectation, and Γ is large, then Φ is much small than Ψ. This would be usefule because we know that under the lemma's assumtpions Ψ does decrease on expectation.

Proof. First, (7) shows the expected increase in Φ is at most

$$\sum_i (p_i(\alpha + S\alpha^2) - \tfrac{\alpha}{n} + S\tfrac{\alpha^2}{n^2})e^{\alpha x_i}$$
$$\leq \sum_{i \leq n/3} (p_i(\alpha + S\alpha^2) - \tfrac{\alpha}{n} + S\tfrac{\alpha^2}{n^2})e^{\alpha x_i} + (\alpha + S\alpha^2) \sum_{i > n/3} p_i e^{\alpha x_i}$$

Recalling that for $i \leq n/3$, $p_i \leq \frac{1-4\epsilon}{n}$ and that $S\alpha < \epsilon$, the first term is at most $-\frac{2\alpha\epsilon}{n}\Phi_{\leq n/3}$. For the second term we again use the fact that for given Φ, $\sum p_i e^{\alpha x_i}$ is maximized when $\mathbf{p}$ is uniform, and bound it by $\frac{2\alpha}{n}\Phi_{>n/3}$. We have:

$$\mathbb{E}[\Delta\Phi | x(t)] \leq -\frac{2\alpha\epsilon}{n}\Phi_{\leq n/3} + \frac{2\alpha}{n}\Phi_{>n/3}$$
$$\leq -\frac{2\alpha\epsilon}{n}\Phi + \frac{3\alpha}{n}\Phi_{>n/3}$$

Thus $\mathbb{E}[\Delta\Phi | x(t)] \geq -\frac{\alpha\epsilon}{n}\Phi$ implies that $\frac{3\alpha}{n}\Phi_{>\frac{n}{3}} \geq \frac{\alpha\epsilon}{n}\Phi$. Or said differently:

$$\Phi \leq \frac{3}{\epsilon}\Phi_{>\frac{n}{3}} \tag{14}$$

Let $B = \sum_i \max(0, x_i) = \frac{1}{2}||x||_1$. Note that $\Phi_{\geq \frac{n}{3}}$ is upper bounded by $\frac{2n}{3}e^{\frac{3\alpha B}{n}}$. Thus

$$\Phi \leq \frac{3}{\epsilon}\Phi_{>\frac{n}{3}} \leq \frac{2n}{\epsilon}e^{\frac{3\alpha B}{n}}. \tag{15}$$

On the other hand, $x_{\frac{3n}{4}} > 0$ implies that $\Psi \geq \frac{n}{4}e^{\frac{4\alpha B}{n}}$.

If $\Phi < \frac{\epsilon}{4}\Psi$, we are already done. Otherwise,

$$\frac{2n}{\epsilon}e^{\frac{3\alpha B}{n}} \geq \Phi \geq \frac{\epsilon}{4}\Psi \geq \frac{\epsilon n}{16}e^{\frac{4\alpha B}{n}}$$

so that $e^{\frac{\alpha B}{n}} \leq \frac{32}{\epsilon^2}$. It follows that

$$\Gamma \leq \tfrac{5}{\epsilon}\Phi \leq \tfrac{40n}{\epsilon^2}(\tfrac{32}{\epsilon})^3 \leq cn.$$

□

Similarly,

Lemma 4.11. *Suppose that $x_{\frac{n}{4}} < 0$ and $\mathbb{E}[\Delta\Psi | x(t)] \geq -\frac{\alpha\epsilon}{4n}\Psi$. Then either $\Psi < \frac{\epsilon}{4}\Phi$ or $\Gamma < cn$ for some $c = poly(\frac{1}{\epsilon})$.*

Proof. First observe that for any $i > \frac{2n}{3}$, $p_i > \frac{1+\epsilon}{n}$ so that $p_i(-\alpha + S\alpha^2) + (\frac{\alpha}{n} + S\frac{\alpha^2}{n^2}) \leq -\frac{\alpha\epsilon}{2n}$. Since $p_i \geq 0$ it holds that $p_i(-\alpha+S\alpha^2)+(\frac{\alpha}{n}+S\frac{\alpha^2}{n^2}) \leq \frac{2\alpha}{n}$ for every i. Using the upper bound from (11) we get

$$\begin{aligned}\mathbb{E}[\Delta\Psi \mid x(t)] &\leq -\frac{\alpha\epsilon}{2n}\Psi_{>\frac{2n}{3}} + \frac{2\alpha}{n}\Psi_{\leq\frac{2n}{3}} \\ &= -\frac{\alpha\epsilon}{2n}\Psi + \frac{4\alpha+\alpha\epsilon}{2n}\Psi_{\leq\frac{2n}{3}} \\ &\leq -\frac{\alpha\epsilon}{2n}\Psi + \frac{3\alpha}{n}\Psi_{\leq\frac{2n}{3}}.\end{aligned}$$

Thus $\mathbb{E}[\Delta\Psi \mid x(t)] \geq -\frac{\alpha\epsilon}{4n}\Psi$ implies that

$$\frac{3\alpha}{n}\Psi_{\leq\frac{2n}{3}} \geq \frac{\alpha\epsilon}{4n}\Psi.$$

Let $B = \sum_i \max(0, x_i) = \frac{1}{2}||x||_1$. Note that $\Psi_{\leq\frac{2n}{3}}$ is upper bounded by $\frac{2n}{3}e^{\frac{3\alpha B}{n}}$. Thus

$$\Psi \leq \frac{12}{\epsilon}\Psi_{\leq\frac{2n}{3}} \leq \frac{8n}{\epsilon}e^{\frac{3\alpha B}{n}}. \tag{16}$$

On the other hand, $x_{\frac{n}{4}} < 0$ implies that $\Phi \geq \frac{n}{4}e^{\frac{4\alpha B}{n}}$.

If $\Psi < \frac{\epsilon}{4}\Phi$, we are already done. Otherwise,

$$\frac{8n}{\epsilon}e^{\frac{3\alpha B}{n}} \geq \Psi \geq \frac{\epsilon}{4}\Phi \geq \frac{n\epsilon}{16}e^{\frac{4\alpha B}{n}}$$

so that $e^{\frac{\alpha B}{n}} \leq \frac{128}{\epsilon^2}$. It follows that

$$\Gamma \leq \frac{5}{\epsilon}\Psi \leq \frac{40n}{\epsilon^2}(\frac{128}{\epsilon})^3 \leq cn.$$

□

We are now ready to prove the supermartingale-type property of Γ.

Theorem 4.12. *Let Γ be as above. Then $\mathbb{E}[\Gamma(t+1) \mid x(t)] \leq (1-\frac{\alpha\epsilon}{4n})\Gamma(t)+c$, for a constant $c = c(\epsilon) = poly(\frac{1}{\epsilon})$.*

Proof. The proof proceeds via a case analysis. In case the conditions, $x_{\frac{n}{4}} \geq 0$ and $x_{\frac{3n}{4}} \leq 0$ hold, we show both Φ and Ψ decrease in expectation. If one of these is violated Lemmas 4.10 and 4.11 come to the rescue.

Case 1: $x_{\frac{n}{4}} \geq 0$ **and** $x_{\frac{3n}{4}} \leq 0$**.** In this case the theorem follows from Lemmas 4.8 and 4.9.

Case 2: $x_{\frac{n}{4}} \geq x_{\frac{3n}{4}} > 0$**.** Intuitively, this means that the allocation is very non symmetric with big holes in the less loaded bins. While Φ may sometimes grow in expectation, we will show that if that happens, then the asymmetry implies that Γ is dominated by Ψ which decreases. Thus the decrease in Ψ offsets the increase in Φ and the expected change in Γ is negative.

Formally, if $\mathbb{E}[\Delta\Phi|x] \leq -\frac{\alpha\epsilon}{4n}\Phi$, Lemma 4.9 implies the result. Otherwise, by Lemma 4.10 there are two subcases:

Case 2.1: $\Phi < \frac{\epsilon}{4}\Psi$**.** In this case, using Lemma 4.9 and Corollary 4.5

$$\mathbb{E}[\Delta\Gamma|x] = \mathbb{E}[\Delta\Phi|x] + \mathbb{E}[\Delta\Psi|x] \leq \frac{2\alpha}{n}\Phi - \frac{\alpha\epsilon}{n}\Psi + 1 \leq -\frac{\alpha\epsilon}{2n}\Psi + 1 \leq -\frac{\alpha\epsilon}{4n}\Gamma + 1$$

Case 2.2: $\Gamma < cn$**.** In this case, Corollaries 4.5 and 4.7 imply that

$$\mathbb{E}[\Delta\Gamma|x] \leq \frac{2\alpha}{n}\Gamma \leq 2c\alpha.$$

On the other hand, $c - \frac{\alpha\epsilon}{4n}\Gamma \geq c(1 - \frac{\alpha\epsilon}{4}) > 2c\alpha$.

Case 3: $x_{\frac{3n}{4}} \leq x_{\frac{n}{4}} < 0$**.** This case is similar to case 2. If $\mathbb{E}[\Delta\Psi|x] \leq -\frac{\alpha\epsilon}{4n}\Psi$, Lemma 4.8 implies the result. Otherwise, by Lemma 4.11 there are two subcases:

Case 3.1: $\Psi < \frac{\epsilon}{4}\Phi$**.** In this case, using Lemma 4.8 and Corollary 4.7, the claim follows.

Case 3.2: $\Gamma < cn$**.** This case is the same as case 2.2. □

Once we have shown that Γ decreases in expectation when large, we can use that to bound the expected value of Γ.

Theorem 4.13. *For any $t \geq 0$, $\mathbb{E}[\Gamma(t)] \leq \frac{4c}{\alpha\epsilon}n$.*

Proof. We show the claim by induction. For $t = 0$, it is trivially true. By Theorem 4.12, we have

$$\begin{aligned}
\mathbb{E}[\Gamma(t+1)] &= \mathbb{E}[\mathbb{E}[\Gamma(t+1) \mid \Gamma(t)]] \\
&\leq \mathbb{E}[(1 - \frac{\alpha\epsilon}{4n})\Gamma(t) + c] \\
&\leq \frac{4c}{\alpha\epsilon}n(1 - \frac{\alpha\epsilon}{4n}) + c \\
&\leq \frac{4c}{\alpha\epsilon}n - c + c
\end{aligned}$$

The claim follows. □

Finally recall that $\texttt{Gap}'(t)$ denotes the additive gap between the maximum bin and the minimum bin, i.e. $x_1(t) - x_n(t)$. We are now ready to prove Theorem 4.3.:

Note that $\Gamma(t) \geq e^{\alpha Gap(t)}$. Since $e^{\alpha x}$ is convex:

$$\begin{aligned}\mathbb{E}[\texttt{Gap}'(t)] &\leq \tfrac{1}{\alpha}\log \mathbb{E}[\Gamma(t)] \\ &\leq \log n/\alpha + \log(4c/\alpha\epsilon)/\alpha \\ &= \log n/\alpha + O(\log(1/\alpha\epsilon)/\alpha).\end{aligned}$$

Similarly, $\Pr[\texttt{Gap}'(t) > 2\log n/\alpha + O(\log(1/\alpha\epsilon)/\alpha)] \leq \Pr[\Gamma(t) \geq n\mathbb{E}[\Gamma(t)]] \leq 1/n$.

4.2 Back to Greedy$[d]$

We are now ready to prove Theorem 4.2. The main idea of the proof is to use Theorem 4.13 to obtain some bounds on the allocation at time $m - n\log n$, and then use the standard induction approach similar to Section 3.1 for the last $n\log n$ balls.

We define the normalized *load vector* $x(t)$ to be an n dimensional vector where $x_i(t)$ is the difference between the load of the i'th bin after tn balls are thrown and the average t, (so that a load of a bin can be negative and $\sum x_i(t) = 0$). This is similar to the definition of load vector in the previous section, only now it would be useful to count time in epochs of n balls each. We also assume as before that the vector is sorted so that $x_1(t) \geq x_2(t) \geq ... \geq x_n(t)$.

The main tool we use is Theorem 4.13 which in our context states that for every $d > 1$ there exist positive constants a and b such that for all n and all t,

$$\mathbb{E}\left[\sum_i \exp\left(a|x_i^t|\right)\right] \leq bn. \tag{17}$$

and in patricular we know that for any t, any $c \geq 0$,

$$\Pr[\texttt{Gap}'(t) \geq \frac{c\log n}{a}] \leq \frac{bn}{n^c} \tag{18}$$

We remark that (18) is tight up to constant factors: in Greedy$[d]$ the lightest bin indeed trails the average by a logarithmic number of balls. The challenge is therefore to use a different technique to "sharpen" the bound on the gap between maximum and average. We do this next by showing that

if the gap is indeed bounded by $\log n$, then after additional $n \log n$ balls are thrown the gap is reduced to $\log \log n$.

The crucial lemma, that we present next, says that if the gap at time t is L, then after throwing another nL balls, the gap becomes $\log \log n + O(\log L)$ with probability close to 1. Roughly speaking, our approach is to apply the lemma twice, first with $L = O(\log n)$ where L is bounded by 17. This reduces the bound to $O(\log \log n)$. A second application of the lemma with $L = O(\log \log n)$ implies Theorem 4.2.

Lemma 4.14. *There is a universal constant $\gamma = \gamma(d)$ such that the following holds: for any t, ℓ, L such that $2^{(d-1)} \leq \ell \leq L \leq \log^2 n$ and $\Pr[\texttt{Gap}(t) \geq L] \leq \frac{1}{2}$,*

$$\Pr\left[\texttt{Gap}(t+L) \geq \frac{\log \log n}{\log d} + \ell + \gamma\right] \leq \Pr[\texttt{Gap}(t) \geq L] + \frac{bL^{\frac{3}{d-1}}}{\exp(a\ell)} + \frac{1}{n^2},$$

where a, b are the constants from (17).

Intuition: We use the induction based technique, similar to Section 3.1. For a specific ball to increase the number of balls in a bin from i to $i+1$, it must be placed in a bin with at least i balls. Under the inductive hypothesis there are at most $\beta_i n$ such bins, so the probability of this happening is at most β_i^d. There are a total of L balls placed, so there are (on expectation) at most $nL\beta_i^d$ bins with load at least $i+1$. Roughly speaking this implies that $\beta_{i+1} \approx L\beta_i^d$. While the β_i's are a function of time, they are monotonically increasing and using the final β_i value would give us an upper bound on the probability of increase. Following this recursion for $\log \log n / \log d$ steps would shrink β to be below $1/n$. The main challenge is to obtain a base case for the induction. In Section 3.1 the base case is implied by the assumption that the number of balls is n. Here, the key observation is that (17) provides us with such a base case, for bins with ℓ more balls than the average in $x(t+L)$. For simplicity, the reader may think of L as $O(\log n)$ and ℓ as $O(\log \log n)$. With these parameters (17) implies that the fraction of bins with load at least $\ell = O(\log \log n)$ (at time $t+L$) is at most $\frac{1}{4 \log n}$, so the β's shrink in each induction step even though $n \log n$ balls are thrown. As mentioned above, we will use the lemma a second time for $L = O(\log \log n)$ and $\ell = O(\log \log \log n)$.

Proof of Lemma 4.14. We sample an allocation $x(t)$ and follow the Markov chain for additional L steps to obtain $x(t+L)$, in other words, an additional nL balls are thrown by the Greedy[d] process. For brevity, we will

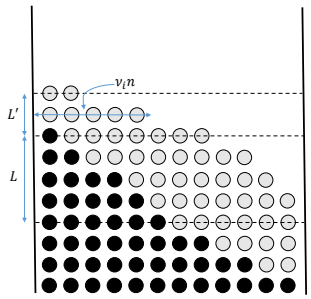

Figure 1: Black balls are in X, nL white balls are thrown to obtain X'

use X, G, X', G' to denote $x(t), \texttt{Gap}(t), x(t+L), \texttt{Gap}(t+L)$ respectively. We condition on $G < L$ and we prove the bound for G'. Let $L' = \log\log n + \ell + \gamma$. Observe that:

$$\Pr[G' \geq L'] \leq \Pr[G' \geq L' \mid G < L] + \Pr[G \geq L] \tag{19}$$

It thus suffices to prove that $\Pr[G' \geq L' \mid G < L] \leq \frac{bL^{\frac{3}{d-1}}}{\exp(a\ell)} + \frac{1}{n^2}$. We do this using a layered induction similar to the one in Section 3.1.

Let ν_i be the fraction of bins with normalized load at least i in X' (i.e. containing $t+L+i$ balls or more), we will define a series of numbers β_i such that $\nu_i \leq \beta_i$ with high probability. To convert an expectation bound to a high probability bound, we will use a Chernoff-Hoeffding tail bound as long as $\beta_i n$ is large enough (at least $\log n$). The case for larger i will be handled separately.

By (17) and along with the assumption $\Pr[G < L] \geq \frac{1}{2}$, Markov's inequality implies that,

$$\Pr\left[\nu_\ell \geq 2L^{-\frac{3}{d-1}} \mid G < L\right] \leq \frac{bL^{\frac{3}{d-1}}}{\exp(a\ell)}. \tag{20}$$

We set $\beta_\ell = 2L^{-\frac{3}{d-1}}$ as the base of the layered and set $\beta_{i+1} = \max(2L\beta_i^d, 18\log n/n)$.

Let $i^* = \ell + \log\log n/\log d + \frac{d+2}{d-1}$.

Lemma 4.15. $\beta_{i^*} = 18\log n/n$.

Proof. Suppose that the truncation does not come into play until i^*. Let ℓ' be such that $\beta_{\ell'} = (2L)^{-\frac{3}{d-1}}$. We first observe that since $\beta_{i+1} \leq \beta_i/2$ it holds that $\ell' \leq \ell + \frac{d+2}{d-1}$. Now, the recurrence

$$\log \beta_{\ell'} = -\tfrac{3}{d-1} \log(2L),$$
$$\log \beta_{i+1} = d \log \beta_i + \log(2L)$$

solves to $\log \beta_{\ell'+k} = \log(2L)(-\frac{3}{d-1} \cdot d^k + \frac{d^k-1}{d-1})$ so that

$$\log \beta_{i^*} = \log \beta_{\ell' + \frac{\log\log n}{\log d}} \leq \log(L)(-2 \log n)$$

This is at most $-2 \log n$ as $\log(2L)/(d-1) \geq 1$ so that $\beta_{i^*} \leq \frac{1}{n^2}$ which is smaller than the truncation threshold, contradicting the assumption. □

The inductive step is encapsulated in the next lemma. The proof is an expectation computation, followed by an application of the Chernoff-Hoeffding bound. Let $B(n, p)$ denote a binomially distributed variable with parameters n and p.

Lemma 4.16. *For $i \in [\ell, i^* - 1]$, it holds that*

$$\Pr[\nu_{i+1} > \beta_{i+1} \mid \nu_i \leq \beta_i, G < L] \leq \frac{1}{n^3}.$$

Proof. For convenience, let the balls existing in X be black, and let the new nL balls thrown be white. We define the *height* of a ball to be the load of the bin in which it was placed relative to X', that is, if the ball was the k'th ball to be placed in the bin, the ball's height is defined to be $k - (t + L)$. Notice that the conditioning that $G < L$ implies that all the black balls have a negative height. We use μ_i to denote the number of white balls with height $\geq i$. For any $i \geq 0$, we have $\nu_i n \leq \mu_i$ and thus it suffices to bound μ_i.

In Greedy[d] , by definition, the probability a ball has height at least $i+1$ is at most ν_i^d which under our conditioning is at most $\beta_i^d \leq \beta_{i+1}/2L$. Since we place nL balls independently, the number of balls with height at least $i + 1$ is dominated by a $B(nL, \beta_{i+1}/L)$ random variable. Chernoff bounds (e.g. Theorem 1.1 in [42]) imply that the probability that $\Pr[B(n, p) \geq 2np] \leq \exp(-np/3)$. Thus

$$\begin{aligned}\Pr[\nu_{i+1} \geq \beta_{i+1} \mid \nu_i \leq \beta_i] &\leq \Pr[B(nL, \beta_{i+1}/2L) \geq \beta_{i+1} n] \\ &\leq \exp(-\beta_{i+1} n/6) \\ &\leq 1/n^3.\end{aligned}$$

since $\beta_{i+1} n \geq 18 \log n$. □

It remains to bound the number of balls with height $\geq i^*$. To this end we condition on $\nu_{i^*} \leq \beta_{i^*}$, and let H be the set of bins of height at least i^* in X'. Once a bin reaches this height, an additional ball falls in it with probability at most $\beta_{i^*}^{d-1}/n \leq 1/n^{1+\epsilon}$ for some $\epsilon > 0$ which depends only on d and large enough n. The probability that any specific bin in H gets at least k balls after reaching height i^* is then at most $\Pr[B(nL, 1/n^{1+\epsilon}) \geq k]$. Recalling that $\Pr[B(n,p) \geq k] \leq \binom{n}{k} p^k \leq (enp/k)^k$. Using this estimate and applying a union bound over the bins in H, we conclude that for $k = 4/\epsilon$

$$\begin{aligned} Pr[\nu_{i^*+k} > 0 \mid \nu_{i^*} \leq \beta_{i^*}, G < L] &\leq 18 \log n \cdot \left(\frac{eL}{kn^{\epsilon}}\right)^k \\ &\leq \frac{1}{2n^2}, \end{aligned} \tag{21}$$

as long as n exceeds an absolute constant n_0. On the other hand, Equation 18 already implies that for $n \leq n_0$, Lemma 4.14 holds with $\gamma = O(\log n_0)$ so that this assumption is without loss of generality.

Finally a union bound using (20) and Lemma 4.16 and (21), we get that

$$\begin{aligned} &\Pr[\nu_{i^*+k} > 0 \mid G < L] \\ &\quad \leq \Pr[\nu_\ell \geq \beta_\ell \mid G < L] + \sum_{i=\ell}^{i^*-1} \Pr[\nu_{i+1} > \beta_{i+1} \mid \nu_i \leq \beta_i, G < L] \\ &\qquad + Pr[\nu_{i^*+k} > 0 \mid \nu_{i^*} \leq \beta_{i^*}, G < L] \\ &\quad \leq \frac{bL^{\frac{3}{d-1}}}{\exp(a\ell)} + \frac{\log\log n}{n^3} + \frac{1}{2n^2} \\ &\quad \leq \frac{bL^{\frac{3}{d-1}}}{\exp(a\ell)} + \frac{1}{n^2}. \end{aligned}$$

This concludes the proof of Lemma 4.14. □

Lemma 4.14 allows us to bound $\Pr[\texttt{Gap}(t+L) \geq \log_d \log n + O(\log L)]$ by $\Pr[\texttt{Gap}(t) \geq L] + \frac{1}{poly(L)}$. Since $\Pr[\texttt{Gap}(t) \geq O(\log n)]$ is small, we can conclude that $\Pr[\texttt{Gap}(t + O(\log n)) \geq O(\log_d \log n)]$ is small. Another application of the lemma, now with $L = O(\log_d \log n)$ then gives that

$$\Pr[\texttt{Gap}(t + O(\log n) + O(\log\log n)) \geq \log_d \log n + O(\log\log\log n)]$$

is small. We formalize these corollaries next.

Corollary 4.17. *There are constants γ, ζ which depend only on d, such that for any $t \geq (4 \log n)/a$, $\Pr[\textit{Gap}(t) \geq \zeta \log\log n + \gamma] \leq \frac{2}{n^2} + \frac{1}{\log^4 n}$.*

Proof. Set $L = 4 \log n/a$, and use 18 to bound $\Pr[\texttt{Gap}(t - L) \geq L] \leq \frac{b}{n^3}$. Set $\ell = \ln(bL^{3/(d-1)} \log^4 n)/a = O_d(\log\log n)$ in Lemma 4.14 to derive the result. □

Corollary 4.18. *There are universal constants γ, α such that for any $t \geq \omega(\log n)$, $\Pr[\textit{Gap}(t) \geq \log_d \log n + \alpha \log\log\log n + \gamma] \leq \frac{3}{n^2} + \frac{1}{\log^4 n} + \frac{1}{(\log\log n)^4}$.*

Proof. Set $L = \zeta \log\log n + \gamma$ and use Corollary 4.17 to bound $\Pr[\texttt{Gap}(t - L) \geq L]$. Set $\ell = \log(bL^{3/(d-1)}(\log\log n)^4)/a = O(\log\log\log n)$ to derive the result. □

This proves that with probability $(1 - o(1))$, the gap is at most $\log_d \log n + o(\log\log n)$. We can also use Lemma 4.14 to bound the expected gap. Towards this end, we prove a slight generalizations of the above corollaries:

Corollary 4.19. *There is a universal constant γ such that for any $k \geq 0$, $t \geq (12 \log n)/a$, $\Pr[\textit{Gap}(t) \geq (5 + \frac{10}{a}) \cdot \log_d \log n + k + \gamma] \leq \frac{2}{n^2} + \frac{\exp(-ak)}{\log^4 n}$.*

Proof. Set $L = 12 \log n/a$, and use (18) to bound $\Pr[\texttt{Gap}(t - L) \geq L]$. Set $\ell = k + \log(16bL^3 \log^4 n)/a$ to derive the result. □

Corollary 4.20. *There are universal constants γ, α such that for any $k \geq 0$, $t \geq \omega(\log n)$, $\Pr[\textit{Gap}(t) \geq \log_d \log n + \alpha \log\log\log n + k + \gamma] \leq \frac{3}{n^2} + \frac{1}{\log^4 n} + \frac{\exp(-ak)}{(\log\log n)^4}$.*

Proof. Set $L = \log(16b(\frac{12 \log n}{a})^3 \log^4 n)/a = \frac{7 \log\log n}{a} + O_{a,b}(1)$ and use Corollary 4.19 with k=0 to bound $\Pr[\texttt{Gap}(t - L) \geq L]$. Set $\ell = k + \log(16bL^3(\log\log n)^4)/a$ to derive the result. □

Using the above results, we can now prove

Corollary 4.21. *There are universal constants γ, α such that for $t \geq \omega(\log n)$ $\mathbb{E}[\textit{Gap}(t)] \leq \log_d \log n + \alpha \log\log\log n + \gamma$.*

Proof. Let $\ell_1 = \log_d \log n + \alpha \log\log\log n + \gamma_1$ for α, γ_1 from Corollary 4.20, and let $\ell_2 = (5 + \frac{10}{a}) \cdot \log_d \log n + \gamma_2$ for γ_2 from Corollary 4.19. Finally, let

$\ell_3 = 12 \log n/a$. We bound

$$\begin{aligned}&\mathbb{E}[\texttt{Gap}(t)]\\ &\leq \ell_1 + \int_{\ell_1}^{\ell_2} \Pr[\texttt{Gap}(t) \geq x]\,\mathrm{d}x + \int_{\ell_2}^{\ell_3} \Pr[\texttt{Gap}(t) \geq x]\,\mathrm{d}x + \int_{\ell_3}^{\infty} \Pr[\texttt{Gap}(t) \geq x]\,\mathrm{d}x\end{aligned}$$

Each of the three integrals are bounded by constants, using Corollaries 4.20 and 4.19 and (18) respectively. □

All that remains to complete the proof of Theorem 4.2 is to show that the lower bound condition on t is unnecessary.

Lemma 4.22. *For $t \geq t'$, $\texttt{Gap}(t')$ is stochastically dominated by $\texttt{Gap}(t)$. Thus $\mathbb{E}[\texttt{Gap}(t')] \leq \mathbb{E}[\texttt{Gap}(t)]$ and for every k, $\Pr[\texttt{Gap}(t') \geq k] \leq \Pr[\texttt{Gap}(t) \geq k]$.*

The proof uses the notion majorization in a standard way and is deferen to Section 4.3

4.2.1 The Left$[d]$ Scheme

Essentially the proof of Theorem 4.2 required two components. The first is the bound on the potential function shown in Theorem 4.13, and the second is an induction based proof for the lightly loaded case shown in Theorem 3.1. This proof structure also applies for the Left$[d]$, extending the bounds to the heavily loaded case as well.

We sketch the argument. The first observation is that Theorem 4.13 indeed holds for Left$[d]$. The reason is that the exponential potential function is Schur-Convex and therefore the theorem holds for any process which is majorized by Greedy$[d]$. A fact that is shown in [12] via a straightforward coupling. See Section 4.3 for a discussion of majorization and coupling.

All that remains is to prove the analog of Lemma 4.16. This follows in a similar manner via the inductive relation expressed in Equation (3).

4.2.2 The Weighted Case

So far we assumed all balls are of uniform weight. Theorem 4.3 however allows for balls to be assigned a weight drawn from a distribution $\mathcal{D}$, as long as $\mathcal{D}$ has a finite exponential moments. It is therefore natural to extend the inductive part of the proof for the weighted case as well. Consider for

example the case where the size of each ball is drawn uniformly from $\{1, 2\}$. Previous techniques such as [12] fail to prove an $O(\log \log n)$ bound in this case, and Theorem 4.3 only shows that $\texttt{Gap} \leq O(\log n)$. In this example, the few modifications that need to be done in order to extend the proof are straight forward. The layered induction argument works as is, with the only change being that we go up in steps of size two instead of one. This shows a bound of $2 \log \log n + O(\log \log \log n)$ for this distribution, which we soon show is tight up to lower order terms.

Generalizing the argument, for a random variable W drawn from a weight distribution $\mathcal{D}$ with a bounded exponential moment generating function, let M be the smallest value such that $\Pr[W \geq M] \leq \frac{1}{n \log n (\log \log n)^5}$ (the constant 5 here is somewhat arbitrary, and will only affect the probability of the gap exceeding the desired bound). Then carrying out a proof analogous to Lemma 4.14, with step size M gives a bound of $M(\log \log n + O(\log \log \log n))$ with probability $(1 - \frac{3}{(\log \log n)^4})$. This follows since by the definition of M, the probability that any of the last $O(n \log n)$ balls exceeds a size of M is $O(\frac{1}{(\log \log n)^5})$, and conditioning on this event the proof goes through unchanged.

Indeed, the lemma is used with $L = O(\log n)$, the base of the induction implies that for $\ell = O(\log \log n)$, the fraction of bins with load at least ℓ is at most $\frac{1}{L^3}$. By the argument in Lemma **??**, no more than $\beta_{i_L+1} n$ balls will fall in bins that already have at least this load. Since we condition on the $O(n \log n)$ white balls being of size at most M, the number of bins of load $\ell + M$ is at most $\beta_{i_L+1} n$. Continuing in this fashion, with step size M in each step of the induction, there are at most $O(\log n)$ bins of load larger than $O(\log \log n) + M \log_2 \log n$. Finally, as before, the argument is completed with an additional overhead of $O(M)$ as each of these bins is unlikely to get more than a constant number of balls. Finally, a second application of the Lemma implies the claimed bound.

It is interesting to instantiate this bound for some specific distributions. For an exponential or a geometric distribution, the gap is $\Theta(\log n)$ and this induction approach will not help us prove a better bound. Consider a half-normal weight distribution with mean 1 (i.e. W is the absolute value of an $N(0, \frac{\pi}{2})$ random variable. Then $M = \sqrt{\pi} erf^{-1}(1 - \frac{1}{n \log n (\log \log n)^5}) = \Theta(\sqrt{\log n})$. This gives an improved bound of $O(\sqrt{\log n} \log \log n)$ instead of $O(\log n)$. On the other hand, as we show in the next section, a lower bound of $\Omega(\sqrt{\log n})$ is easily proved.

Similarly, if the weight distribution is uniform in $[a, b]$, normalizing the expectation to 1 makes $b = 2 - a \leq 2$. An upper bound of $b \log \log n \leq$

$2\log\log n$ follows immediately.

We note that Lemma 4.22 does not hold when balls are weighted (c.f [99],[13]). As a result this proof leaves a "hole" between n and $n\log n$. It proves the bound on the gap when $O(n)$ or $\Omega(n\log n)$ balls are thrown but does not cover for example $\Theta(n\sqrt{\log n})$ balls.

4.2.3 Lower Bounds

If weights are drawn uniformly from $\{1,2\}$ one might hope the maximum load to be $3/2\log\log n + O(1)$. It is true that $n/2$ balls of weight 2 already cause a gap of $2\log\log n$ but one hopes that the balls of weight 1 would reduce this gap. Our first lower bound shows that this intuition is not correct and that the maximum load is indeed $2\log\log -O(1)$.

Theorem 4.23. *Suppose the weight distribution $\mathcal{D}$ satisfies $\Pr[W \geq s] \geq \epsilon$ for some $s \geq 1, \epsilon > 0$ and $\mathbb{E}[W] = 1$, where s, ϵ are independent of n. Then, for large enough n, for every $m \geq n/\epsilon$, the gap of Greedy[d] is at least $s(\log\log n/\log d) - O(s)$ with constant probability.*

A similar lower bound is proven in [9] for the case $m = n$ and uniform weights. In the uniform case, majorization (similar to Lemma 4.22) extends the lower bound to any $m > n$. The same could not be said in the weighted case. For the $m = n$ case the weighted case is almost as simple as a variable change in the proof of [9]. The proof below extends the bound to all $m \geq n$. The main idea, similarly to the upper bound, is that (17) impleis a base case for an inductive argument.

Proof. It is convenient to think of time m as time 0 and count both load and time with respect to the m'th ball, so when we say a bin has load i in time t it actually means it has load $w(m)/n + i$ at time $m + t$, where $w(m)$ is the total weight of the first m balls. The bound will be proven for time $m + n/\epsilon$ which is time n/ϵ in our notation. Intuitively, in this amount of time we will see about n balls of weight at least s which would cause the maximum load to increase by $s(\log\log n - O(1))$. The average however would increase only by $O(\frac{1}{\epsilon}) = O(s)$, hence the gap would be at least $s\log\log n - O(s)$.

We follow the notation set in [74], with appropriate changes. The variable $\nu_j(t)$ indicates the number of bins with load in $[(j-1)s, \infty)$ at time t. We will set a series of numbers γ_i and times t_i (to be specified later) and an event $\mathcal{F}_i := \{\nu_i(t_i) \geq \gamma_i\}$. For the base case of the induction we set $\gamma_0 = n/\log^2 n$ and $t_0 = 0$. We observe that Theorem **??** implies that for

large enough n, $\Pr[\nu_0(0) \geq \gamma_0] \geq 1 - 1/n^2$, so $\mathcal{F}_0$ occurs with high probability. Indeed Theorem **??** implies that for the normalized load vector, $|X^t|_\infty \leq c \log n$ for an absolute constant c. If half the X_i^t's are at least $-s$, we are already done. If not then then $\sum_{i:X_i^t<-s} |X_i^t|$ is at least $\frac{ns}{2}$. Thus the sum $\sum_{i:X_i^t\geq 0} |X_i^t| = \sum_{i:X_i^t<0} |X_i^t| \geq \frac{ns}{2}$. The bound on $|X^t|_\infty$ then implies that at least $ns/c \log n$ X_i^t's are non-negative. Since $s \geq 1$, the base case is proved.

Our goal is to show that $\Pr[\mathcal{F}_{i+1}|\mathcal{F}_i]$ is large. To this end, we define $t_i = (1 - 2^{-i})\frac{n}{\epsilon}$ and the range $R_i := \left[(1 - 2^{-i})\frac{n}{\epsilon}, (1 - 2^{-(i+1)})\frac{n}{\epsilon}\right]$. Finally fix an $i > 0$ and for $t \in R_i$ define the binary random variable

$Z_t = 1$ iff ball t pushes load of a bin above is or $\nu_{(i+1)}(t-1) \geq \gamma_{i+1}$.

As long as $\nu_{(i+1)}(t-1) < \gamma_{i+1}$ it holds that for $Z_t = 1$ it suffices that a ball of weight at least s is placed in a bin of load $h \in [s(i-1), si)$. Conditioned on $\mathcal{F}_i$, the probability of that is at least $\epsilon\left((\frac{\gamma_i}{n})^d - (\frac{\gamma_{i+1}}{n})^d\right) \geq \frac{\epsilon\gamma_i^d}{2n^d}$ since we will set $\gamma_{i+1} \leq \gamma_i/2$. Denote $p_i := \frac{\epsilon\gamma_i^d}{2n^d}$ and by $B(n,p)$ a variable distributed according to the Binomial distribution. We have: $\Pr\left[\sum_{i\in R_i} Z_i \leq k \mid \mathcal{F}_i\right] \leq \Pr\left[B\left(\frac{n}{\epsilon 2^{i+1}}, p_i\right) \leq k\right]$. We continue exactly as in [**?**] by choosing $\gamma_{i+1} = \frac{\gamma_i^d}{2^{i+3}n^{d-1}}$. Now Chernoff bounds imply that as long as $\frac{np_i}{\epsilon 2^{i+1}} \geq 17 \log n$ it holds that $\Pr\left[B\left(\frac{n}{\epsilon 2^{i+1}}, p_i\right) \leq \gamma_{i+1}\right] = o(1/n^2)$. The tail inequality holds as long as $i \leq \log\log n/\log d - O(1)$, at which point the load had increased by $s(\log\log n/\log d) - O(s)$. The average increased by at most $4/\epsilon \leq 4s$ with probability $3/4$, and the theorem follows. □

We note that the uniform distribution on $\{1, 2\}$ (when normalized by a factor of $\frac{2}{3}$) satisfies the conditions of this Theorem with $s = 2, \epsilon = \frac{1}{2}$. Thus the gap is $2\log\log n - O(1)$.

Another, rather trivial lower bound applies to distributions with heavier tails.

Theorem 4.24. *Let $\mathcal{D}$ be a weight distribution with $\mathbb{E}_{W\sim\mathcal{D}}[W] = 1$. Let M be such that $\Pr_{W\sim\mathcal{W}}[W \geq M] \geq \frac{1}{n}$. Then for any allocation scheme, the gap is at least $M - O(1)$ with constant probability.*

Proof. After throwing n balls, the probability that we do not see a ball of weight M or more is at most $(1 - \frac{1}{n})^n \leq \frac{1}{2}$. Moreover, by Markov's, the average is at most 4 except with probability $\frac{1}{4}$. Thus with probability at least $\frac{1}{4}$, the maximum is at least M and the average is at most 4. □

Note that the theorem implies an $\Omega(\log n)$ lower bound for an exponential distribution, and an $\Omega(\sqrt{\log n})$ lower bound for the half normal distribution.

4.3 The Power of Majorization

In some cases it is possible to say that one process dominates another. Establishing a partial order of stochastic dominance is a powerful tool which extends the analysis for a larger set of processes.

Definition 4. *Consider two vectores $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$. For every j let $x_{[j]}$ be the j'th largest entry of x. We say that x is* majorized *by y denoted by $x \preceq y$ if for every i,*

$$\sum_{j \le i} x_{[j]} \le \sum_{j \le i} y_{[j]}.$$

One can observe that the majorization relation is transitive.

Definition 5. *A function $\gamma : A \to \mathbb{R}$, where $A \subset \mathbb{R}^n$ is said to be* Schur-convex *if $x \preceq y$ implies $\gamma(x) \le \gamma(y)$.*

It is known that every symmetric and convex function is also Schur-Convex (see page 258 in [95]), so in particular the function that tracks the maximum value in the vector, $\gamma(\cdot) = ||\cdot||_\infty$, is Schur-convex, and so is the potential function Γ from Theorem 4.13.

Definition 6. *Let x, y be two distributions*[1] *over $\mathbb{R}^n$. We say that x is* stochastically majorized *by y, written as $x \preceq y$ if for every Schur-convex function γ it holds that $\mathbb{E}[\gamma(x)] \le \mathbb{E}[\gamma(y)]$.*

A standard technique for showing stochastic majorization is via *coupling*. A coupling of x and y is a distribution $z = (z_1, z_2)$ such that the marginal distribution $(z_1, \cdot)$ is identical to x, and the marginal distribution $(\cdot, z_2)$ is identical to y. There could be arbitrary dependencies in the joint distribution z. The following lemma encapsulates a simple observation that demonstrates the power of coupling.

Lemma 4.25. *Let x and y be two distrubtions over $\mathbb{R}^n$. If there exists a coupling $z = (z_1, z_2)$ of x, y such that $z_1 \preceq z_2$ then $x \preceq y$.*

Proof. We assume for simplicity the distributions are discrete. Let γ be some Schur-convex function. Since z is a valid coupling we have $\mathbb{E}[\gamma(x)] = \mathbb{E}_z[\gamma(z_1)] = \sum_z \gamma(z_1) \Pr[z]$ and similarly $\mathbb{E}[y] = \sum_z \gamma(z_2) \Pr[z]$. The lemma follows since γ is Schur-convex so for every term $z = (z_1, z_2)$ it holds that $\gamma(z_1) \le \gamma(z_2)$. □

[1] With a slight abuse of notation we conflate a distribution with a sample taken from it.

The following lemma is taken from [9] and serves as the main tool for estabilishing the majorization relation.

Lemma 4.26. *Let x, y be two vectors in $\mathbb{R}^n$ such that $x \preceq y$ and assume that both are sorted in decreasing order. Let e_i denote the i'th unit vector, i.e. e_i has 1 in the i'th coordinate and zero everywhere else. If $i \geq j$ the $x + e_i \preceq y + e_j$.*

Proof. Observe that if $i > j$, then $y_i \leq y_j$ and $y + e_i$ could be obtained from $y + e_j$ by moving a unit from a large component to a smaller component, therefore $y + e_i \preceq y + e_j$. It is now sufficient to show that $x + e_i \preceq y + e_i$. Let $S_k(x)$ be the sum of the k largest components of x. Notice that

$$S_k(x) \leq S_k(x + e_i) \leq S_k(x) + 1 \tag{22}$$

By assumption for all k, $S_k(x) \leq S_k(y)$. Fix some k. Now, if $S_k(x) < S_k(y)$ then (22) implies that $S_k(x + e_i) \leq S_k(y + e_i)$. Assume that $S_k(x) = S_k(y)$. There are 3 cases:

Case 1. $i \leq k$. Then

$$S_k(x + e_i) = S_k(x) + 1 = S_k(y) + 1 = S_k(y + e_i).$$

Case 2. $i > k$. Then $x_i \leq x_k$

Case 2.a. $x_i < x_k$. Since $x_k \geq x_i + 1$ it follows from (22) that $S_k(x) = S_k(x + e_i)$ and therefore

$$S_k(x + e_i) = S_k(x) = S_k(y) \leq S_k(y + e_i).$$

Case 2.b. $x_k = x_{k+1} = \ldots = x_i$. Observe that since $S_{k-1}(x) \leq S_{k-1}(y)$, $S_k(x) = S_k(y)$ and $S_{k+1}(x) \leq S_{k+1}(y)$ we have that $y_k \leq x_k$ and $y_{k+1} \geq x_{k+1}$. Since y and x are sorted we have

$$y_k \geq y_{k+1} \geq x_{k+1} = x_k \geq y_k.$$

We conlude that $x_k = y_k = x_{k+1} = y_{k+1}$ and thus $S_{k+1}(x) = S_{k+1}(y)$. Repeating the argument we obtain $x_k = y_k = x_i = y_i$ and therefore

$$S_k(y + e_i) = S_k(y) + 1 = S_k(x) + 1 = S_k(x + e_i).$$

□

The following lemma establishes a way of comparing different processes in the case of uniform weights. It states that a majorization relationship across the allocation probabilities implies stochastic majorization across the respective precesses.

Theorem 4.27. *Let p and q be two probability vectors of two generalized placement processes (see Definition 1). Let $x(t)$ and $y(t)$ be the load vectores of these processes respectively at time t, and assume all balls are of weight 1. If $q \preceq p$ then for every t, $x(t) \preceq y(t)$.*

Proof. The proof uses Lemma 4.25 in an inductive way. For the base case, both $x(0)$ and $y(0)$ are $\{0\}^n$, so the lemma holds trivialy.

By the inductive hypothesis we have a coupling $(z_1(t-1), z_2(t-1))$ of $x(t-1), y(t-1)$ such that $z_1(t-1) \preceq z_2(t-1)$. Consider the following coupling.

1. Pick a number α uniformly at random in $[0, 1)$.
2. Assume p and q are sorted in decreasing order. Let i be such that $\sum_{\ell \le i-1} p_\ell \le \alpha < \sum_{\ell \le i} p_\ell$. Similarly, let j be such that $\sum_{\ell \le j-1} q_\ell \le \alpha < \sum_{\ell \le j} q_\ell$.
3. Obtain $x(t)$ by placing a ball in the i'th bin and $y(t)$ by placing a ball in the j'th bin.

It is straightforward to verify that this is indeed a valid coupling. It remains to show that throughout the coupling $x(t)$ is majorized by $y(t)$. At time t a ball is put in the i'th bin of $x(t-1)$ and in the j'th bin of $y(t-1)$. However since $q \preceq p$ it must be that $j \le i$. The Theorem now follows by Lemma 4.26. □

In the remainder of the section we show some surprising examples where majorization and Theorem 4.27 are used to bound the load of various processes that are hard to analyze directly.

4.3.1 Greedy[d] with Non-uniform Sampling probability

Implementing Greedy[d] requires the ability to sample uniformly from the set of n bins d times. In practice however obtaining a perfectly uniform sampled bin is often not achievable. Consider for instance distributed hash tables or key-value stores which are important and widely used applications. Often they imploy a scheme called *consistent hashing* [56]. The main idea is that both servers and items (key-value pairs) are hashed to the same key

space. An item is assigned to the server closest to it in the key space. This general idea is prevalent both in the theory of distributed hash tables (c.f. [98], [82]), and also in practical and widely used systems such as Cassandra [23] and many more. Note that this scheme effectevily partitions the key space across servers. The probablity a server is sampled is proportional to the fraction of key space in its partition. Since these systems are large and dynamic it is unreasonable to expect the key space to be partitioned equally across all servers. As a result the distribution in which servers are sampled is not uniform, and the unlucky servers end up with a larger fraction of the items.

A natural way to increase the balance across servers is to implement Greedy[d] , that is, to hash each item to several locations and place it in the least loaded server. Let $\mathcal{Q} = q_1, \ldots, q_n$ be some distribution over the n bins. We define Greedy[$d, \mathcal{Q}$] to be the allocation process whereby a ball is put in the least loaded of d bins sampled according to $\mathcal{Q}$. In particualr, Greedy[d] is a special case where $\mathcal{Q}$ is the unifrom distribution.

It is not apriori obvious whether Greedy[$d, \mathcal{Q}$] would be effective. In [20] it was shown that when $m = n$ Greedy[$d, \mathcal{Q}$] guarantees a maximal load of $O(\log \log n)$ even when $d = 2$ and $\mathcal{Q}$ is fairly imbalanced. Things become trickier when $m >> n$. Consider for instance the case where $d = 2$ and $q_1 = 2/\sqrt{n}$. The probablity that bin 1 obtains a ball is at least $(2/\sqrt{n})^2 = 4/n$, so this bin is expected to store at least 4 times the average, so the gap grows linearly with m. Next we follow the arguments of [105] which show that increasing d by a little would alleviate the problem.

Assume $t-1$ balls had already been placed by the Greedy[$d, \mathcal{Q}$] process. Let $p(d, \mathcal{Q}, t)$ be the probablity vector that characterized Greedy[$d, \mathcal{Q}$] for the t'th ball (see Definition 1). In other words $p(d, \mathcal{Q}, t)_i$ is the probability Greedy[$d, \mathcal{Q}$] places ball t in the i'th bin. Note that it is important to index it by t as the value of the vector depends on the specific allocation of balls. Similarly, let $p(d,\mathcal{U})$ be the placement vector that characterizes Greedy[d] . Recall, that $p(d,\mathcal{U})_i := (\frac{i}{n})^d - (\frac{i-1}{n})^d$.

Lemma 4.28. *Let $0 < \alpha < 1 < \beta$, $\epsilon > 0$ be such that for all i, $q_i \in [\frac{\alpha}{n}, \frac{\beta}{n}]$ and $d \geq (1+\epsilon)\frac{\ln\left(\frac{\beta-\alpha}{1-\alpha}\right)}{\ln\left(\frac{\beta-\alpha}{\beta-\beta\alpha}\right)}$. Then for all t, $p(1+\epsilon,\mathcal{U}) \preceq p(d, \mathcal{Q}, t)$.*

The Lemma is proven via a straight forward calculation, see [105] for details. Togethor with Theorem 4.27 and Theorem 4.1 the Lemma immediately implies:

Corollary 4.29. *For α, β, ϵ as above, after m balls are placed using Greedy[$d, \mathcal{Q}$]*

, *with probability* $1-o(1/n)$ *the maximum load would be at most* $\frac{m}{n}+\frac{\ln\ln n}{\ln(1+\epsilon)}+O(1)$

≪Put a numerical example –Udi≫ UW

We note that the converse also holds. If $d \leq (1-\epsilon)\frac{\ln\left(\frac{\beta-\alpha}{1-\alpha}\right)}{\ln\left(\frac{\beta-\alpha}{\beta-\beta\alpha}\right)}$ then there is a distribution $\mathcal{Q}$ with each $q_i \in [\alpha/n, \beta/n]$ for which the maximal load is expected to be $(1+\Omega(1))m/n$.

4.3.2 The $(1+\beta)$ process

The $(1+\beta)$ process is a general allocation processes parameterized by $\beta \in [0,1]$. Each time a ball is placed, with probability β, Greedy[2] is used, and with probability $1-\beta$ the ball is placed using Greedy[1], i.e. simply placed in a random bin. In other words, the process is characterized by the allocation vector p where $p_i = \frac{\beta(2i-1)}{n^2} + \frac{1-\beta}{n}$. The process turns out to be a useful process to compare against using majorization, but it is also interesting in its own right. Consider for instance the case where the cost assigned to querying the load of the bins is relatively high. This scenario may occur in a distributed storage system in which a front-end server places data items in back-end servers. Using Greedy[2] requires querying two servers for their load, possibly locking them until the data item is placed. Moreover, a lookup for the item results in two queries. In the $(1+\beta)$ process with probability $1-\beta$ the assignment is done without performing this query, and the expected lookup cost is only $(1+\beta)$. The question of course is how much imbalance is introduced due to this modified procedure. In [72] it was suggested to study the $(1+\beta)$-process in the context of queueing theory. This process may also serve to model the case where all balls perform Greed[2], but some fraction of them are 'misinformed', say by an unreliable load reporting mechanism, and make the wrong decision.

As a general placement process, the $(1+\beta)$ process satisfies assumption (5) and (6) with $\beta = \Theta(\epsilon)$. Recall that $\texttt{Gap}(m)$ denotes the additive difference between the maximal bin and the average load, and that $\texttt{Gap}'(m)$ denotes the additive difference between the maximal and minimal bins. Theorem 4.3 implies:

Corollary 4.30. *For the* $(1+\beta)$ *process,* $\mathbb{E}[\textit{Gap}(m)] \leq \mathbb{E}[\textit{Gap}'(m)] \leq O\left(\frac{\log n}{\beta}\right)$.

We later show that this bound is tight up to constants for β bounded away from 1. Thus, $\texttt{Gap}(m)$ has three regimes as a function of β. When $\beta = 0$ we are looking at Greedy[(1)], the gap is $\Theta(\sqrt{\frac{m\log n}{n}})$ which goes to

infinity with m. When $0 < \beta < 1$ the gap is $\Theta((\log n)/\beta)$ independent of m, and when $\beta = 1$ we are looking at Greedy[2] and the gap decreases further to $\log \log n$. We note that $\texttt{Gap}'(m) = \Omega(\log n)$ also for Greedy[d] .

With Corollary 4.30 at our disposal we consider new allocation schemes. As a first example consider the bias-away-from-max and bias-towards-min processes, defined as follows:

$$p_i^{away\_from\_max} = \begin{cases} \frac{1-\eta}{n} & i = 1 \\ \frac{1}{n} + \frac{\eta}{n(n-1)} & \text{otherwise} \end{cases}$$

and

$$p_i^{towards\_min} = \begin{cases} \frac{1+\eta}{n} & i = n \\ \frac{1}{n} - \frac{\eta}{n(n-1)} & \text{otherwise} \end{cases}$$

It is easy to check that both these processes are majorized by the $(1+\frac{\eta}{4n})$-choice process so by Theorem 4.27 the expected gap for these processes is bounded by $O(n \log(n/\eta)/\eta)$. It's remarkable that even a small bias away from the max or towards the min suffices for the gap to stay bounded independent of m. It's worth noting that it is not obvious how to analyze these processes directly or apply the techniques in [12, 99].

4.3.3 Graphical Processes

Let $\mathcal{B}$ be the set of all subsets of bins. Assume we have some fixed distribution τ over $\mathcal{B}$. A ball is placed by sampling a $S \in \mathcal{B}$ according to τ and placing the ball in the least loaded of the bins in S. We assume for explicitness that ties are broken according to some fixed ordering of the bins. Observe that Greedy[d] is a special case where $\tau(S) = 1/n^d$ for all S of cardinality at most d. Similarly Left[d] is captured by setting $\tau(S) = (d/n)^d$ for every $|S| = d$ which contains one bin from every set. A similar model was suggested in [58] where sets are limited to be of cardinality 2, so the bins could be associated with nodes of a graph, and the sets with its edges. They show that if the graph is regular and dense enough then a layered induction approach could be used to prove bounds in the spirit of Theorem 3.1. A related model is also presented in [54] where it is shown that if all sets are of logarithmic size and are approximately regular then a constant load is achieved with $m = n$. Here we follow a more general framework taken from [89].

The main difficulty with analyzing graphical processes is that generally they *cannot* be characterized by a probability vector. In particular if a bin has the smallest load of all bins which share a set with it, then the probability

it receives a ball is zero, even if its load is above the global average. We do not know how to analyze these processes directly. Our approach is to show that in the unweighted case the gap of a graphical process is bounded by the gap of a $(1+\beta)$ process for an appropriately chosen β. We show a majorization relationship that holds for every specific allocation of balls into bins.

We say that τ is ϵ-*expanding* if every $S \subset B$ of cardinality at most $|S| \leq \frac{n}{2}$ has $\sum_{A \subseteq S} \tau(A) \leq \frac{(1-\epsilon)|S|}{n}$ and $\sum_{A: A \cap S \neq \emptyset} \tau(A) \geq \frac{(1+\epsilon)|S|}{n}$.

Theorem 4.31. *If τ is ϵ-expanding, then the expected gap between the heaviest and lightest bins after throwing m balls is $O(\frac{\log n}{\epsilon})$.*

Proof. Let $\mathbf{p}(t)$ be the probability vector characterizing the process after placing t balls. Note that $\mathbf{p}(t)$ may indeed depend upon the specific allocation at time t, this dependency however plays no role in the argument below. Let $\mathbf{q}$ denote the probability vector of the $1+\beta$ process. We show that if $\epsilon \geq \beta$ then $\mathbf{p} \preceq \mathbf{q}$, and so the theorem follows directly from Theorem 4.27.

Let S_k be the set of k lightest nodes (breaking ties according to the tie-breaking rule). We calculate the probability P_k that a ball falls in S_k. A ball falls in S_k if and only if the sampled set has a non-empty intersection with S_k. The expansion of τ implies that $P_k \geq (1+\epsilon)k/n$ when $k \leq n/2$ and $P_k \geq 1 - \frac{(1-\epsilon)(n-k)}{n} = \frac{k+\epsilon(n-k)}{n}$ when $k > n/2$. Now let Q_k denote the probability a ball falls in the k lightest bins in the $1+\beta$ process, so $Q_k = (1-\beta)k/n + \beta(1-(1-k/n)^2) = \frac{k}{n}(1+\beta-\frac{\beta k}{n})$. One can readily verify that if $\epsilon \geq \beta$ then $P_k \geq Q_k$ for all k. Indeed, if $k \leq n/2$ it holds that

$$\frac{(1+\epsilon)k}{n} \geq \frac{k}{n}(1+\epsilon-\frac{\epsilon k}{n})$$

and if $k > n/2$ we have that

$$\frac{k+\epsilon(n-k)}{n} \geq \frac{k}{n}(1+\epsilon-\frac{\epsilon k}{n})$$

□

The theorem implies explicit bounds for some interesting processes.

Expander Graphs: Assume we have a graph over the sets of bins, and τ samples uniformly an edge in the graph (that is the model considered in [58]). Observe that a constant degree regular expander satisfies the condition of the theorem. Such an expander has only $O(n)$ edges and still obtains a logarithmic gap. In particular, if an n node d regular graph has the property

that every set S of cardinality at most $n/2$ is adjacent to at least $(1+\epsilon)|S|$, then a uniform measure over its edges is ϵ/d expanding.

Rings: As a second example assume the bins are indexed by $0, 1, ..., n-1$. The assignment algorithm samples uniformly $i \in [n]$ and puts the ball in the least loaded of the bins $i, i+1, ..., i+k-1$, where operations are done modulo n. This algorithm is natural in the common scenario where the bins are represented by adjacent locations in memory (or network), and it is desirable to have an access pattern that preserves locality. It is also common in key-value stores which use consistent hashing, such as Amazon's Dynamo [29] where the range $i, ..., i+k$ is used for replication and may store multiple copies of the object. The theorem implies that even such a moderate attempt at load balancing can bound the gap by $O(\frac{n \log n}{k})$. To the best of our knowledge it was previously unknown that the gap does not diverge. There is no reason to believe this bound is tight, in fact, simulations seem to suggest the true gap to be much smaller. Improving over Theorem 4.31 for this case is an intruiging and challenging open problem.

4.4 A Lower Bound

Theorem 4.3 is tight, even for the case of Greedy[d] . This does not contradict Theorem 4.1 since it bounds the additive gap between the maximum and the minimum, while Theorem 4.1 bounds the gap between the maximum and the average. To see this note that if the placement algorithm samples sets S of size bounded by d then after throwing $\Omega(\frac{n \log n}{d})$ with constant probability there would be a bin that was untouched by any of the sets and thus must remain empty. The gap between the average and the minimum is therefore $\Omega(\frac{\log n}{d})$, even in the case of Greedy[d] .

For the $(1+\beta)$ scheme we show a tight lower bound also for the gap between maximum and average. Consider the $(1+\beta)$-choice process, and recall that $\epsilon \in \Theta(\beta)$. Throw $\frac{an \log n}{\beta^2}$ balls into n bins using the $(1+\beta)$ choice process, where a is to be defined later. The average load is thus $\frac{a \log n}{\beta^2}$. The expected number of balls thrown using one choice is $\frac{an(1-\beta) \log n}{\beta^2}$. Raab and Steger [93] show that the load of the most loaded bin when throwing $cn \log n$ balls into n bins using 1-choice is at least $(c + (\sqrt{c}/10)) \log n$. Plugging in $c = \frac{a(1-\beta)}{\beta^2}$, we get that the number of balls in the most loaded bin is at least

$$\left(\frac{a(1-\beta)}{\beta^2} + \sqrt{\frac{a(1-\beta)}{100\beta^2}} \right) \log n = \left(\frac{a}{\beta^2} + \frac{\sqrt{a(1-\beta)} - 10a}{10\beta} \right) \log n.$$

It is easy to see that for $a = \frac{(1-\beta)}{200}$, this is $\Omega((1-\beta)\log n/\beta)$ more than the average.

4.5 Adaptive Schemes

Adaptive schemes are ones where the number of random bins picked for a ball is not fixed apriori, as in Greedy[d] but rather depends upon the load observed. The scheme *ADAPTIVE* was proposed in [14] following the work in [26]: m balls are placed sequentially. For each ball i repeatidly sample bins uniformly and independently, until a bin with load at most $\lceil i/n \rceil$ is found. Ball i is then placed in that bin. Clearly the maximum load is at most $\lceil m/n \rceil + 1$. The question is how many probes are performed by the algorithm. It is shown that the expected total number of probes *ADAPTIVE* performs over the placement of m balls is $O(m)$. Note that this algorithm is not suitable as a basis for a dictionary because there is no criteria for when an item is *not* in the bins, so the worst case time for a (unsuccessful) lookup is n.

5 Dictionaries

Dictionaries are an abstract data structure that stores a set of *(key, value)* pairs. Typically the assumption is that all keys are distinct. The data structure supports the operations `Insert(`*`key,value`*`)`, `Lookup(`*`key`*`)`, and `Delete(`*`key`*`)`. In practice the values may be stored adjacently to the keys, or if they are big, the values would be pointers to larger objects. While these differences have a big performance impact in practice they are orthogonal to the design of the dictionary itself, so here we focus on the keys themselves. The keys are drawn from a universe $\mathcal{U}$. If the dictionary is to store n items we assume that $|\mathcal{U}| \leq poly(n)$, so in particular keys are assumed to be binary strings of length $O(\log n)$. The justification of this assumption is derived from the observation that if the universe is larger it could be hashed down by a pairwise independent function to a polynomially small universe with the probability of collisions over a set of size n being polynomially small. This approach is known as 'collapsing the universe' and we do not discuss it further, see [101],[67] for some approaches. We operate in the *word RAM* model, where is is assumed that each memory word is comprised of $O(\log n)$ bits and is big enough to store a full key. Thus, in this terminology, to store all keys we need linear space. Note that a trivial lower bound for recognizing whether a key is in the data structure is $\log \binom{\mathcal{U}}{n}$ which is $\Omega(n \log n)$ bits, or

$\Omega(n)$ memory words when $\mathcal{U}$ is polynomial in n. In most of what follows we also keep the random hashing assumption, which means we assume we can compute a fully random mapping from $\mathcal{U} \to T$ where T is a much smaller domain, typically $O(n)$ in size.

In the previous sections we presented the $d-$way chained hash table and saw how multiple choice schemes underly the analysis. In practice, the operation of allocating new memory is quite costly. When assessing the space requirement of a dictionary we therefore often assume that all the space is allocated in advance. When treated this way, the hash table based on Left$[d]$ must allocate large enough buckets upfront and uses $O(n \log\log n/d)$ space. Insertion time is $O(d)$ (or $O(\log\log n)$ depending on implementation details) and lookup time is $O(\log\log n)$ in the worse case. If $\log\log n$ hash functions could be evaluated in parallel than the lookup time could be improved to $O(1)$ as was shown in [18]. Could this be improved?

The answer is yes. In a classic work Fredman et. al. [49] presented a static construction, that is, a dictionary where all items are given up front and it supports only lookups. The data structure has linear space and constant time lookups. It was made dynamic in [35] where the insertions and deletions are in amortized expected constant time. Further improvements given in [34] and [32] where all operations are constant time with high probability. We will see a construction with similar properties at Section 5.2.

5.1 Cuckoo Hashing

A natural way to improve over d-way chaining and constructions based on Greedy$[d]$ is to allow the items to *move* between their designated bins. *Cuckoo Hashing* [86] is a data structure that does just that. The items are stored in two tables T_1, T_2, each containing $(1+\epsilon)n$ slots where $\epsilon > 0$ is independent of n. A slot is simply enough space to store an item from the universe U and is the unit of space for this discussion. So the space is $O(n)$, in fact just slightly more than $2n$. Cuckoo Hashing uses two hash functions $h_1, h_2 : U \to [(1+\epsilon)n]$. As usual, for now the hash functions are assumed to be fully random. To store a set $S \subseteq U$, each item $x \in S$ is stored either in $T_1[h_1(x)]$ or in $T_2[h_2(x)]$. Since each item may reside in one of only two locations, maintaining this invariant guarantees that the lookup requires at most two computations of the hash functions and two memory probes. We present a modification to Cuckoo Hashing, suggested in [61] where on top of the tables there is also a constant size array called the *stash*. The invariant is modified so that each item may also reside in the stash. Since the stash is constant sized, the lookup time is still bounded by $O(1)$.

When inserting x the first priority is to place it either in T_1 or T_2, and not the stash. If one of its two designated locations are available, x is placed and the insertion succeeds. Of course, it may be the case that both locations are full. In that case x is inserted in $T_1[h_1(x)]$ anyhow, evicting the current occupant of that slot, lets call it y. Now y is inserted at its second viable location at table T_2, if that location is occupied, that item is evicted and inserted at table T_1, possibly evicting a different item and so on. The algorithm continues to iterate that way until either an empty slot is found or until it reaches a certain predefined bound on the number of evictions (which will turn out to be $O(\log n)$). When this bound is reached the insertion to the tables themselves had not succeeded so the algorithms attempts to insert the currently evicted item in the stash. If the stash is also full then the insertion procedure failed. The action taken upon an insertion failure is application dependent and outside our scope. Typically it is to rehash all elements with different hash functions, but in some cases it may be preferable to just drop an element from the dictionary. The pseudo code for the insertion algorithm is below.

Algorithm 1. *Insertion in Cuckoo tables with a stash*

```
1:  procedure INSERT(x)
2:      nestless:= x;
3:      i := 1;
4:      loop maxloop times
5:          swap(nestless,T_i[h_i(nestless)]);
6:          if nestless = nill then return;
7:          i ← 3 − i;
8:      if stash not full then
9:          add x to stash
10:     else return fail
```

end

The running time of the insertion is bounded by the parameter `maxloop`. The two most important things to reason about are the *expected* running time of the insertion algorithm and the probability in which the insertion algorithm fails. These issues are addressed in the following theorem.

Theorem 5.1. *When inserting a set of n elements into two tables of size $(1+\epsilon)n$ each and a stash of size s, and setting `maxloop` $= O(s \log_{1+\epsilon} n)$, the*

expected running time of the Insert procedure is $O(1)$ *and the probability of a failed insertion is* $O(n^{-(s+1)})$ *where* s *is the size of the stash.*

The remainder of the section is dedicated to the proof of this theorem. While there are several approaches, we follow closely the proof in [8]. A short description of alternative approaches could be found in Section 5.1.1. The starting point of all approaches is showing a connection between the failure probability of the insertion algorithm and static structural properties of the 'cuckoo graph'.

Definition 7. *The* Cuckoo Graph $G(S, h_1, h_2)$ *(denoted* $G(S)$ *if context is clear) is a bipartite multigraph where each side consists of* $(1+\epsilon)n$ *nodes, labeled* $1...(1+\epsilon)n$*. For each* $x \in S$ *the pair* $(h_1(x), h_2(x))$ *is an edge for a total of* n *edges.*

A connected component of a graph is unicycle if it contains exactly one cycle. Trivially, a connected component is unicycle if and only if it has the same number of edges and nodes.

Consider an element x and let S be the set of all elements placed in the two tables when x is inserted. Clearly, a necessary condition for the insertion of x to avoid the use of the stash is that the connected component of x in $G(S, h_1, h_2)$ is either a tree or a unicycle. Otherwise the number of elements (including x) is greater than then the number of available slots in the tables and the stash must be used. It turns out that the converse is also true and if the connected component has at most one cycle the insertion procedure would place the item in the tables. Further, the running time of the insertion algorithm is tied to the size and shape of the connected components in the cuckoo graphs. The following lemma states that it is sufficient to set `maxloop` to be either twice the size of a connected component or three times the size of its longest path.

Lemma 5.2. *Let* $c(x)$ *denote the connected component of* x *in the cuckoo graph. If* $c(x)$ *has at most one cycle then there is a* $t < \infty$ *such that the insertion of* x *succeeds after* t *iterations, without using the stash, i.e. the insertion algorithm returned in Line* (6)*. Further,* t *is at most twice the number of nodes of* $c(x)$*, and* $c(x)$ *contains a path of length* $t/3$*.*

The proof is taken from [30].

Proof. Consider the walk W in the cuckoo graph that corresponds to the table cells visited during the insertion of x. The first thing to observe is that W cannot be trapped in a single cycle. To see this let (u, v) be the first

edge of a cycle in W, so the edge (u, v) corresponds to some item y, at first y resides in the slot associated with u, and then is moved by the insertion algorithm to v. Once the cycle returns u there are two options. Either the slot u is empty, in which case the insertion procedure ends, or there is some item placed in u. This item however cannot be y since y now resides in v. It follows that the next edge in W is not part of the cycle.

We now take a closer look at the walk W. If W has no repeated cells then its length is at most $|c(x)|$ as required. Otherwise, let u be the first vertex repeated in W. So, the cuckoo graph contains a cycle C which contains u. The discussion above implies that the edge following u's second occurrence does not belong to C. Now consider the walk W' which is the suffix of W starting after the first occurrence of u in W. Suppose there is a vertex that is repeated in W', let w be the first such vertex. If w is not in C then w belongs to a cycle different than C contradicting the assumption that $c(x)$ is unicyclic. If $w \in C$ there are two cases. In the first case $w = u$. This implies that the edge following u's second occurrence belongs to a cycle different than C. If $u \neq w$ then the path between u's second occurrence and w belongs to a cycle different than C, again, contradicting the assumption that C is the only cycle in the component. Therefore the walk W' does not contain repeated vertices. By construction the walk $W \setminus W'$ also does not contain repeated vertices. We conclude that $t \leq 2|c(x)|$ as required. Further, the occurrences of u partitions W to at most 3 paths completing the proof of the lemma.

□

While the running time of the algorithm depends upon the size of the connected components. The failure probability of the algorithm depends on the their density as captured by the following concept.

Definition 8. *The* excess *of a graph G, denoted as $ex(G)$ is the minimum number of edges one has to remove from G so that all connected components of the remaining graph are either trees or unicycles.*

The following characterization of excess turns out be useful.

Lemma 5.3. *The* cyclolamatic *number of a graph, denoted by $\gamma(G)$ is the minimal number of edges one has to remove from G to obtain a graph with no cycles. Let $\zeta(G)$ denote the number of connected component of G with at least one cycle. Then, $ex(G) = \gamma(G) - \zeta(G)$.*

Proof. Let T be a component of G which contains a cycle. We first claim that $\mathrm{ex}(T) + 1 = \gamma(T)$.

$\geq$: By definition, there is a set of S of $\mathrm{ex}(T)$ edges, the removal of which leaves at most one cycle. Since T contains a cycle, the minimality of S implies that $T \setminus S$ also contains exactly one cycle. Removing one additional edge from the remaining cycle creates an acyclic graph.

$\leq$: By definition there is a set of edges S such that $|S| = \gamma(T)$ and $T \setminus S$ is acyclic. Further, the minimality of S implies that $T \setminus S$ is connected. Now we can add back one more edge from S creating a component with at most one cycle.

The lemma follows by summing up the excess across the cyclic components of G. □

It turns out that the excess of the cuckoo graph is a crucial parameter determining the success of the insertion procedure.

Lemma 5.4. *Given a set of keys S and two hash functions h_1, h_2 and assuming `maxloop` is large enough as in Lemma 5.2. If the Insertion algorithm inserts the S items sequentially and places s items in the stash then $s = ex(G(S, h_1, h_2))$.*

Proof. A connected graph over k nodes has exactly one cycle if and only if it has k edges. Therefore, if a set S' is stored only in the tables T_1, T_2, it must be the case that every connected component of $G(S')$ has at most one cycle. Otherwise, a connected component with more than one cycle would have more elements than slots.

$\geq$: Assume that $T \subset S$ is stored in the stash, and $S' = S \setminus T$ is stored in the two tables. The discussion above implies that every connected component of $G(S')$ is either acyclic or unicyclic, which by the definition of excess means that $s = |T| \geq \mathrm{ex}(G)$.

$\leq$: We use induction on the cardinality of S. If S is empty then both the excess and the stash size are 0. Assume that a set of keys S had been inserted, a set of keys T had been put in the stash. By the induction hypothesis $|T| \leq \mathrm{ex}(S)$. We now insert a new element $y \in U \setminus S$. We proceed by case analysis:

Case 1: The item y is inserted in the tables without placing an item in the stash. So $|T| \leq \mathrm{ex}(S) \leq \mathrm{ex}(S \cup y)$.

Case 2: The insertion procedure places some item in the stash. We need to show that the excess of the graph increased. Let $S' = S \setminus T$. We know by Lemma 5.2 that this implies that $G(S' \cup \{y\})$ must contain

a connected component that is neither a tree nor a unicyclic. Since $\mathrm{ex}(G(S')) = 0$ it must be the edge corresponding to y that makes the difference. Now we use Lemma 5.3. If y is a cycle edge in $G(S')$ it is also a cycle edge in $G(S)$, the cyclomatic number of its component increased by one while the number of cyclic components remained unchanged. If y connects to cyclic component in $G(S)$ then the cyclomatic number remained unchanged while the number of cyclic components decreased. Either way $\mathrm{ex}(S \cup y) = \mathrm{ex}(S) + 1$ and the Lemma follows.

□

Lemma 5.4 establishes the excess of the cuckoo graph as the key structural property that determines the success of the insertion procedure. In the following we show that when the hash functions are chosen randomly, it is highly unlikely that the excess is large.

Theorem 5.5. *Let $\epsilon > 0$ and $s \geq 1$. For $n > 1$, $m = (1+\epsilon)n$ let G be the cuckoo graph constructed by a set $S \subseteq U$ of size n, and two hash functions $h_1, h_2 : U \to [m]$ chosen uniformly at random from all possible such functions. Then:*

$$\Pr\left[ex(G) \geq s\right] = O(n^{-s}) \tag{23}$$

Proof. The proof is by a counting argument. We start by defining the object of interest.

Definition 9. *A graph is said to be an* excess$-s$ core graph *if its excess is exactly s, it is leafless and every connected component has at least two cycles.*

Lemma 5.6. *An excess$-s$ core graph with t edges has $t - s$ nodes.*

Proof. We first prove the lemma for the case the graph is connected. By definition there are s cycle edges whose removal leave behind a unicycle connected graph. This graph has $t - s$ edges, and therefore $t - s$ nodes. If the graph has k connected components each with t_i edges and excess s_i then the number of nodes is $\sum(t_i - s_i) = t - s$. □

We count the number of unlabeled excess-s core graphs with t edges.

Lemma 5.7. *The number of non isomorphic unlabeled excess$-s$ core graphs with t edges is $t^{O(s)}$*

Proof. We first claim that the number of trees over k nodes, that have ℓ leaves is bounded by $k^{2\ell}$. To see this we use induction on ℓ. For $\ell = 2$ there is exactly one tree with two leaves: a path of length k. Now, for $\ell \geq 3$ a tree with ℓ leaves is composed of a path of length at most k connected to a tree with $\ell - 1$ leaves. If t is the length of that path then there are at most $k - t$ nodes for this path to connect to. The number of trees is therefore at most

$$\sum_{t=1}^{k} (k-t)^{2(\ell-1)}(k-t) \leq k \cdot k^{2\ell-1} = k^{2\ell}$$

Consider now an excess$-s$ core graph with t edges which is connected. Each such graph could be built by taking a tree with $t - s - 1$ edges and at most $2s + 2$ leaves, and then adding $s + 1$ cycle edges. There are at most t^{2s+2} ways of adding these edges. We conclude that the number of these graphs is bounded by $t^{2s+2} \cdot t^{2s+2} \leq t^{8s}$ as $s \geq 1$.

Finally consider an excess$-s$ core graph with k components. We show by induction that the number of such graphs is bounded by $t^{(8s+2k)}$. The base case is shown above. For the inductive step observe that every such graph is created from one component with $t' < t$ edges and excess $s' < s$, and the remaining $k-1$ components. The number of such graphs is therefore bounded by

$$\sum_{t'<t} \sum_{s'<s} t'^{5s'} \cdot (t-t')^{5(s-s')+2(k-1)} \leq ts \cdot t^{5s+2(k-1)} \leq t^{5s+2k}$$

Since each connected component has at least two cycles, each component contributes at least one edge to the excess. Therfore $k \leq s$ and the lemma follows. □

Every graph G with $\mathrm{ex}(G) \geq s \geq 1$ contains an excess$-s$ core as a subgraph. To see this remove all cycle edges until the excess is exactly s. Then remove leaf nodes until there are none. At this point all components which were trees have been eliminated, and unicyclic component are reduced to simple cycles. Remove components that are simple cycles, and what remains is the excess core graph.

Now that we know how many different core graphs there could be we need to count the number of possible labelings a core graph can get as part of a cuckoo graph. Recall that the cuckoo graph is bipartite with $m = (1+\epsilon)n$ nodes on each side.

Fix an unlabeled bipartite excess$-s$ core graph G with t edges and k components, and fix some $T \subseteq U$ so that $|T| = t$. First, for each (bipartite)

component we need to decide which part is matched with which part of the cuckoo graph, there are $2^k < 2^s$ ways of doing that. We know the number of nodes in G is $t - s$, so there are at most m^{t-s} ways of assigning labels from $[m]$ to the nodes. Finally, there are $t!$ ways of assigning the elements of T to the edges. To conclude, the number of bipartite excess$-s$ core graphs where each node is assigned to a node from the bipartite cuckoo graph, and where the t edges are labeled with distinct elements from T is at most

$$t! \cdot 2^s \cdot m^{t-s} \cdot t^{O(s)}$$

If G with such a labeling is fixed, and the hash functions h_1, h_2 are chosen uniformly at random, then the probability that all edges $(h_1(x), h_2(x)), x \in T$ match the labeling is $1/m^{2t}$.

Let p_T be the probability that the cuckoo graph $G(T, h_1, h_2)$ has excess at least s. By the union bound

$$\begin{aligned}
\sum_{T \subseteq S} p_T &\leq \sum_{t \leq n} \binom{n}{t} \frac{t! \cdot 2^s \cdot m^{t-s} \cdot t^{O(s)}}{m^{2t}} \\
&= \frac{2^s}{m^s} \sum_{t \leq n} \binom{n}{t} \frac{t! \cdot t^{O(s)}}{m^t} \\
&\leq \frac{2^s}{m^s} \sum_{t \leq n} \frac{n^t \cdot t^{O(s)}}{m^t} \\
&\leq \frac{1}{n^s} \cdot \left(\frac{2}{1+\epsilon}\right)^s \cdot \sum_{t \leq n} \frac{t^{O(s)}}{(1+\epsilon)^t} = O\left(\frac{1}{n^s}\right)
\end{aligned}$$

This completes the proof of Theorem 5.5. □

Finally we need to bound the probability a failure is triggered despite the excess being small. This can only happen if `maxloop` is too small.

Lemma 5.8. *Assume a set S of n items is inserted, where the size of each table is $m = (1+\epsilon)n$. Let u, v be two nodes, then probability that there is a simple path from u to v in $G(S)$ of length ℓ is at most $\frac{1}{n(1+\epsilon)^\ell}$*

Proof. The proof is by a counting argument. We count the number of possible paths between u and v. There are m ways to pick the first edge, at most m to pick the second, and so on. The ℓ'th edge must end in v so in total there are at most $m^{\ell-1}$ possible simple paths. There are $\binom{n}{\ell}$ ways to choose a set of ℓ items from S, then there $\ell!$ ways of ordering them, and a

probability of $1/m^{2\ell}$ that they are associated with a given path. All in all, the probability is bounded by

$$\binom{n}{\ell} \cdot m^{\ell-1} \cdot \ell! \cdot \frac{1}{m^{2\ell}} \leq \frac{n^\ell}{m^{\ell+1}} \leq \frac{1}{n(1+\epsilon)^\ell}$$

□

As in Lemma 5.2, let $c(x)$ denote the component of item x in the cuckoo graph. Let y be an edge associated with another item from S. Clearly $y \in c(x)$ iff there is a path connecting $h_1(y)$ and $h_2(x)$. The probability for this according to lemma 5.8 is at most $\sum_\ell \frac{1}{n(1+\epsilon)^\ell} = O(1/n(1+\epsilon))$. We conclude that the expected number of edges in $c(x)$ is $O(1/(1+\epsilon))$, so the by Lemma 5.2 so is the expected running time of the insertion algorithm. Further, by setting $\ell = O(s \log_{1+\epsilon} n)$ we see that with probability $O(1/n^s)$ there is no path of length ℓ in the graph.

Combined with Theorem 5.5 this completes the proof of Theorem 5.1.

Deleting with a stash Deletions are simple with cuckoo hashing without a stash. Removing an item x requires nothing more simply deleting it from its location in the tables. Once a stash is used it might be the case that a deletion of x would allow an item currently in the stash to be placed in the tables. If that item stays in the stash it may cause the stash to become too big. A simple solution to try to insert all items in the stash upon every deletion. This would solve the problem but would cause deletions to take logarithmic time in the worst case w.h.p.. An alternative approach is to try to insert all items of the stash after every insert operation. Deletion now still takes $O(1)$ time in the worst case, and since with high probability the stash contains only $O(1)$ items (and most likely zero), the insert operation is affected by a constant factor.

5.1.1 Alternative Proof Approaches

One advantage of the combinatorial nature of the proof above, it that it allows for further analysis with respect to specific and explicit hash families. In fact, that was the main motivation of the work in [8]. The original cuckoo hashing paper [86] also took a similar approach. A different approach is to look at the problem from a random graph theory perspective. This obscures some of the combinatorics but places the problem within the larger context of random processes. In that approach we observe that the assumption the hash functions are fully random implies that the cuckoo graph is sampled via

a standard model of random graph, where m edges are sampled uniformly over n nodes. Further, in the relevant parameter setting, this distribution is in the the sub critical phase (i.e. before large connected components appear). This domain is widely studied in the literature and a number of tools (such as Poisson approximations, branching processes etc) have been used to identify structural properties of the components. A fairly straightforward application of these techniques imply the result above. This approach was taken in [61].

In [41], [64] they use the assumption of full randomness to prove tight bounds on the failure probability and on the total construction time of the dictionary.

5.2 Some Interesting Variations

The simplicity of cuckoo hashing leads to some interesting variations, we sketch two of these.

De-amortized cuckoo hashing Cuckoo hashing guarantees a worst case constant lookup time and deletion time. The insert operation takes constant time only on expectation and $O(\log n)$ with hight probability. A natural goal is to improve upon this so that all operations take $O(1)$ in the worst cast. The question on whether such a construction is possible is remarkably open and probably one of the most intriguing problems in the field. A slightly weaker objective is to have all operations take $O(1)$ time with high probability. More formally, we say a series of operations is n-bounded if at any given time at most n items exist in the dictionary. Let $p(\cdot)$ be a polynomial, our goal is to construct a dictionary so that for every $p(n)$ operations which are n-bounded, the worst case time per operations over the series is $O(1)$ with probability $1 - 1/p(n)$.

In [34] it is was first shown how to construct a dictionary with this property. We sketch here a construction with similar guarantees based on Cuckoo hashing. It is much simpler and arguably has smaller leading constants. It is taken from [6] later extended in [7] to a succinct data structure.

The main idea is to de-amortize the insertion by using a searchable queue. The approach was first suggested in [60] in a slightly different variant and without analysis. The construction is parameterized by an integer L which depends on $p(\cdot)$. The idea is very simple, on top of the tables T_0, T_1 the construction also maintains a queue Q. This immediately implies that the lookup and delete procedures need to inspect not only the tables and the stash but the queue as well. We will see that the queue will have at most $O(\log n)$ elements in it. The first step is therefore to construct a queue

that supports constant time lookup operations. It is not hard to see that randomly hashing a linked list into a dictionary with n^δ space has this property. Armed with such a queue the question is which items to put in the queue and how to perform the insert operation. Upon in insertion of an item x the pair $(x, 0)$ is placed at $Q.Tail$. The zero tag is a single bit indicates that when it is x's turn to be inserted into the tables, the first location to be tried is the one in T_0. Subsequently a pair (y, b) is removed from $Q.head$, where y is the item and b is the bit which indicates where to try to insert the item. The insert operation now performs at most L moves, dequeuing a new item whenever and item is inserted successfully into the tables. Note that the number of operations may be less than L if the queue is empty. At the end of the L operations the current nestled item is stored, along with the bit indicating where to continue the insert at the head of the queue. If an item closes a second cycle in its connected component it cannot be inserted and is pushed to the back of the queue, effectively conflating the stash with queue.

The performance of the scheme depends upon the queue containing at most $O(\log n)$ items at any point in time. We sketch the argument that this holds with high probability. There are two reasons an item is put in the queue, the first is because an item closes a second cycle in its connected component. We have already seen something similar in Theorem 5.1. A slight modification shows that with probability $1 - 1/p(n)$ there would be at most $O(1)$ of those items in the queue throughout the sequence.

The second reason an item could be inserted in to the queue is simply because L operations are not sufficient to place it in the tables. For an item x let $C(x)$ be its connected component in the cuckoo graph upon insertion. We have seen that the time to insert an item x is $O(|C(x)|)$. It follows that if there are k items in the queue, it must be the case that the sum of sizes of their connected components is at least Lk. For a single item x we have already seen in Lemma 5.8 a bound on $|C(x)|$. The lemma could be rephrased so that $|C(x)|$ is dominated by a geometric variable. In [6] they generalize the analysis for a *set* of connected components and prove the following:

Lemma 5.9. *For a cuckoo hashing scheme with two fully random hash functions and two tables of size* $(1+\epsilon)n$*, any constant* $c_1 > 0$ *and any integer* $T \leq \log n$ *there exists a constant* c_2*, such that for any set of elements* S *of*

size n and a subset $x_1, \ldots, x_T \in S$ it holds that

$$\Pr\left[\sum_{i=1}^{T} |C(x_i)| \geq c_2 T\right] \leq \exp(-c_1 T)$$

where $C(x)$ is the connected component of x in the cuckoo graph determined by S and the two hash functions.

In other words, the sum of sizes of the connected components of T items, concentrate more or less like the sum of T geometric variables. In our setting this implies that with probability $1 - 1/p(n)$ the size of the queue would not exceed $O(\log n)$ as required.

History independent hash tables The notion of history independent [81],[71] arises in the setting where the data structure designer is concerned with an adversary that has access to the actual memory representation of the data structure, and the goal is to hide all information other than the one exposed via the data structure's interface. In the context of dictionaries the goal is that the memory representation would not reveal any information concerning the order in which the items were inserted or whether there were items that were inserted and later removed. This situation may arise for instance in the context of voting machines. Here we are concerned with a variant called *Strong History Independence* that states that the memory representation has a *canonical form*. In other words, it is completely determined by the set currently represented in the dictionary and a fixed amount of initial randomness. An example of a strongly history independent implementation of a dictionary is by keeping all items in a sorted array. The problem with this solution of course is that the time complexity of operations is high. Next we will see a simple and efficient construction based on Cuckoo hashing. For more details see [80]. A different construction based on Linear Probing and that has a weaker notion of history independence could be found in [55].

Our goal therefore is to describe the memory layout of the cuckoo hashing tables and stash after each operation. An item x could potentially be placed in either $T_0[h_0(x)]$ or in $T_1[h_1(x)]$. To achieve strong history independence there has to be a deterministic rule that assigns one of these locations to x. The rule will be based on the shape and the members of the x's component in the cuckoo graph. Note that the cuckoo graph is fully determined by the set of items S and the initial randomness encapsulated in the two hash functions. We now describe these rules. We then show that these rules could

be followed in a rather straight forward way. Let C be a component of the cuckoo graph. There are 3 cases:

C **is a tree:** The number of edges (items) is one less than the number of slots (nodes), so exactly one of the nodes remains vacant. Further, the choice of the vacant node fully determines the placement of all the items of the component and any one of the nodes could potentially serve as the vacant one. Any arbitrary deterministic rule would work but to be specific we determine that the node in C which is in table T_0 and of smallest index is the one to be left vacant.

C **is unicyclic:** Here all the nodes of the component must receive an item. The only variance in placement arises by the placement of the items that lie in the unique cycle of the component. There are two ways to place those items, and we should pick one deterministically. We decide (arbitrarily) that from the items that form the cycle, the item with the smallest key is placed in table T_0.

C **has more than one cycle:** Here the first decision to make is which items to place in the stash. Any deterministic procedure would do. For instance, iteratively place in the stash the item with smallest key that corresponds to a cyclic edge. This is done until the component is unicyclic, at which point the previous case applies.

Upon insertion these invariants could be maintained by scanning the connected component in the cuckoo graph, i.e. in time $O(|C|)$ which is asymptotically similar to the standard cuckoo hashing dictionary. To see that this is possible note that in the first case the item only needs to locate the vacant location in its component, and in the second and third invariants the insertion process needs to locate the cycles in the component.

Performing the Delete operation is slightly more subtle. It could be the case that after the deletion a simple of the cuckoo graph starting from the deleted item will not reach the location that (say) should now be vacant. To fix this we have all the items of the connected component also hold pointers and form a cyclic link list. This list could be traversed instead of the cuckoo graph itself. This reduces the space utilization of the data structure to 0.25 under the conservative assumption that pointers take the same space as keys. The missing details could be found in [80].

5.3 Generalized Cuckoo Hashing and k-Orientability

There are two natural ways to generalize Cuckoo Hashing. The first is to increase the number of hash function used from 2 to a general $d > 1$. This is called the "d-ary cuckoo hashing scheme" and was first studied in [45]. The second is to increase the capacity of a memory location (bin) so that it can store more than one item. This is sometimes refered as "blocked cuckoo-hashing" and was first studied in [39], though the case the capacity is 2 was examined in [87]. We remark that it is also possible to have the bins overlap. This was shown in [65] to carry some advantages, but henceforth we assume bins are distinct. These schemes could of course be combined, hence we define the (d, k)-cuckoo scheme as one that uses d hash functions and a capacity of k in each bin. In this terminology the standard cuckoo hashing scheme described previously is the $(2, 1)$-scheme. The goal of these schemes is to increase the space utilization of the data structure. While in the vanila cuckoo hashing $(2 + \epsilon)m$ slots are needed to store m items, so the space utilization is at most 0.5. In these schemes the number of slots can be much closer to m and space utilization gets closer to 1. There are two main questions that needs to be addressed. The first is how to insert an item. In the $(2, 1)$ an item that is evacuated from its slot has exactly one other potential location, hence the insertion algorithm is self-prescribed. Generally though an item which was evacuated may evacuate one of $dk - 1$ other items and the insertion algorithm needs to specify which one of those to move. Second, it is desirable to quantify the tradeoff between d, k and the actual space utilization obtained by the data structure.

5.3.1 Space Unitilization

Given a (d, k)-cuckoo scheme, let $A_{m,n}$ be the event it is possible to place m items in the n bins (assuming the hash functions are fully random). Note that in this notation the total number of memory slots is nk. A constant $c_{d,k}$ is a *threshold* for the (d, k)-cuckoo placement scheme if:

$$\lim_{n\to\infty} \Pr[A_{\lfloor cn \rfloor, n}] = \begin{cases} 0 \text{ if } c < c_{d,k} \\ 1 \text{ if } c > c_{d,k} \end{cases}$$

In this terminology the space utilization of the scheme is $c_{d,k}/k$ and we have already seen that $c_{2,1} = 0.5$. We note that it is not trivial a-priori that a threshold exists.

An alternative way to phrase the problem is as a *hypergraph orientability* problem. A k-orienting of a hypergraph is an assignment of each edge to

one of its vertices so that each vertex is assigned at most k edges. In this terminology the vertices represent the bins, the edges represent the balls and the orientation of the edge indicates the assignment of the ball. Now the offline (d, k) scheme is tantamount to a k-orienting of a hypergraph, where each edge is of degree d, that is, spans d nodes. Let $\mathcal{H}_{n,m,d}$ be all hypergraphs with n nodes and m edges of degree d. The problem of assigning m balls to n bins via a d, k-cuckoo scheme is equivalent (assuming the hash functions are fully random) to the probability a uniformly sampled member of $\mathcal{H}_{n,m,d}$ has a k-orientation.

The existence and characterization of thresholds was gradually discovered in many papers [96], [21], [44], [33], [52], [47], we refrain from describing the contribution of each one. Finally, a general solution is given in [46] and [66], and asymptotically also in [53].

Theorem 5.10. *For interegers $d \geq 3$ and $k \geq 1$, let ξ be the unique solution of the equation*

$$dk = \frac{\xi Q(\xi, k)}{Q(\xi, k+1)}, \text{ where } Q(x, y) = 1 - e^{-x} \sum_{j<y} \frac{x^j}{j!}$$

and set

$$c_{d,k} = \frac{\xi}{dQ(\xi, k)^{d-1}}$$

then

$$\lim_{n\to\infty} \Pr[H_{n,\lfloor cn \rfloor,d} \text{ is } k\text{-orientable }] = \begin{cases} 0 & \text{if } c < c_{d,k} \\ 1 & \text{if } c > c_{d,k} \end{cases}$$

In the table below we compute a few of the implied space utilizations for the (d, k) cuckoo hashing schemes, up to a third digit rounding. In practice an increase in d and k are not equivalent. Increasing d requires an additional computation of hash function and one more random memory probe which is likely to be a cache miss. On the other hand, a moderate increase in k may come with almost no cost as all if the items in the bucket share the same cache line. Thus, an appealing option in practice is setting $d = 2$ and $k = 4$ for a 0.96 memory utilization.

$d\backslash k$	1	2	3	4
2	0.5	0.87	0.94	0.96
3	0.9178	0.97	0.98	0.999
4	0.9768	0.9982	0.9998	0.9999

In a further generalization proposed in [53] and [66] they consider the case where each hypergraph edge needs to be assigned to $\ell \geq 1$ of its vertices. In data structure terminology this is equivalent to the requirement that each item is replicated ℓ times, each replica to be placed in a different memory locations. In this case as well, for every d, k and $1 \leq \ell < d$ a threshold exists and memory utilization approaches 1 quickly as d, k increase.

It turns out the threshold for the cuckoo scheme (or for orientability) is tightly related to that of cores. A k-core of a hypergraph is a maximum subgraph where each node has degree at least k. The *density* of a graph is the maximum ratio of edges to nodes taken over all induced subgraphs. The following two theorems appear in [46] and establish the two bounds needed to show that $c_{d,k}$ is a threshold. They generalize nicely the results for the vanila version of cuckoo hashing. In that case we were concerned with cyclic components which correspond to $2-$cores.

Theorem 5.11. *If $c > c_{d,k}$, then with probability $1 - o(1)$, the $(k+1)$-core of a hypergraph uniformly sampled from $\mathcal{H}_{\lfloor cn \rfloor, n, d}$ has density greater than k.*

Note that when the density is greater than k the pigeonhole principle implies there cannot be a k orientation. Alternately, in the data structure language, this implies that there is a set of t bins that receive more than kt balls, and therefore there cannot be a placement of these balls respecting a limit of k on the capacity of each bin.

Theorem 5.12. *If $c < c_{d,k}$, then with probability $1 - o(1)$, all subgraphs of a hypergraph uniformly sampled from $\mathcal{H}_{\lfloor cn \rfloor, n, d}$ have density smaller than k.*

Consider a bipartite graph with the m items on the left and the nk memory slots on the right, and place an edge whenever a memory slot is a valid placement for an item. Note that Hall's theorem implies that if the density of the hypergraph is smaller than k then there is a left perfect matching that matches all the m items.

Theorems 5.11 and 5.12 highlight a fundamental difference between the $(2, 1)$ case and the remaining cases. In the $(2, 1)$ case, while below the threshold all connected components are either trees or unicycles, and we know they are small in size as well. the components are small in size as well. The fact the subgraphs we care about are the components themselves simplifies the analysis and allows for the simple constructions of explicit hash functions and the history independent scheme. On the other hand when $d \geq 3$ or $k \geq 2$ there objects of interest are cores and subgraphs,

and these may lie within a large connected component. This explains the difficulty in analyzing insertion algorithms and explicit hash functions.

5.3.2 Insertion Algorithms

The previous section considered the space utilization without specifying a solution to the computational problem of actually finding the allocation. Clearly in the offline case this is simply a matching problem. The interesting questions are whether it is possible to find a good *online* algorithm where the items arrive one by one, and whether there exists a linear time offline algorithm.

The $(d, 1)$ scheme was first suggested in [45] under the name $d-$ary cuckoo hashing. In that work (which predated Theorem 5.10) they show that for a space utilization of $1 - \epsilon$ it is sufficient to set $d = O(\ln(1/\epsilon))$. Theorem 5.10 implies this is tight up to multiplicative constants. They suggested the following natural insertion algorithm: consider a bipartite graph where the left side is associated with items and the right side with memory locations. We place an edge (u, b) if b is one of u's d valid memory locations. Further, if u is placed in b we orient the edge towards u, otherwise it is oriented towards b. An online insertion algorithm can scan this graph starting from u until it finds an empty memory location. A natural algorithm to scan this graph is a BFS, where one of the advantages is that the actual evictions (which presumable are a costly operation) would occur along a shortest path. The paper proves that the *expected* number of memory probes the BFS algorithm performs is some constant exponential in $1/\epsilon$.

In [39] they propose and analyze the $(2, k)$-scheme, under the name blocked-cuckoo-hashing. Similarly, they identify the threshold and show that setting $k \geq 1 + \frac{\ln(1/\epsilon)}{1-\ln 2}$ is sufficient for a space utilization of $1/(1 + \epsilon)$. While asymptotically this is similar to that of the $(d, 1)$ scheme, as we mentioned before, in practice an increase in k is less costly than an increase of d. They show that if k is larger, but still $O(\ln(1/\epsilon))$ then a BFS insertion algorithm runs in constant time on expectation, again with the constant exponential in $1/\epsilon$.

A natural algorithm for both cases is the *Random Walk* algorithm. In that algorithm when an item needs to be evacuated, a random item out of the dk items is picked and is moved to a *random* location out of the remaining $dk - 1$ locations. This is tantamount to random walk on the graph as described previously. This algorithm was proposed already in [45] and [39] but eluded analysis for a long time. The first analysis for the $(d, 1)$ scheme was given in [50] where is is shown that the expected insertion time

of the random walk algorithm is at most $\log^{2+\delta(d)} n$ where $\delta(d) \to 0$ as d increases. The bound holds however only if d is sufficiently large, even compared to the space utilization of the scheme. This result was improved in two ways. In [48] essentially the same bound is shown, but this time it is shown to hold when d is down all the way to the space utilization threshold. In [51] it is shown that the expected insertion time is *constant*, but d has to be in the order of $\frac{\log 1/\epsilon}{\epsilon}$, while recall that the threshold for a space utilization of $1-\epsilon$ is roughly $\log 1/\epsilon$. They also show that the *maximum* insertion time is bounded by $\log^{O(1)} n$ w.h.p.

A third interesting online insertion algorithm was suggested in [59] which he calls the *Local Search* algorithm, which is also aimed at the $(d, 1)$ case. In this algorithm each bin b is given an integer label $L(b)$ initially set to zero. Given an item x let $b_1, \ldots, b_d$ be its d possible locations. Let b_i be a bin with minimal label among those d bins (ties broken arbitrarily). The algorithm proceeds by doing the following two steps:

1. $L(b_i) \leftarrow \min_{j \neq i} L(b_j) + 1$. In words: The label of $L(b_i)$ is updated to be one more than the smallest label of the other $d-1$ labels.

2. The item x is placed in bin b_i. If there is an item already there it is evacuated from the bin and reinserted using the same procedure.

The paper proves two statements. The fist is that if m items could be placed in n bins then no label would ever exceed $n-1$. This means that the algorithm always finds a placement if one exists and has a worst case running time of $O(n^2)$. Further, a label that reaches a value of n is a certificate that no placement exists. Secondly, the paper proves that if the d locations are chosen randomly then with high probability it takes $O(n)$ time to place m items, even as m gets close to $c_{3,1}n$.

There is no known high probability bound on the insertion time or a single item, but experiments reported in [59] suggest the maximum to be logarithmic, and significantly smaller than the maximum of a random walk algorithm. Further, the total insertion time of items is a full order of magnitude smaller than random walk.

The algorithm comes with the additional cost of storing the labels. It is shown that with high probability the labels do not exceed $O(\log n)$ thus this algorithm presents good trade-offs for most applications.

5.4 Linear Probing

Perhaps the most common hashing technique is Linear Probing. It was invented in the 50's and was first analyzed by Knuth in 1963 in one of the

true classics of algorithm analysis [63]. In this scheme an array T of length n should store m items taken from universe U. The load m/n is denoted by α. The scheme uses a hash function $h : U \to [n]$ which for now is assumed to be fully random. Item x is placed in the first free cell of the list $\{T[h(x)], T[h(x)+1], T[h(x)+2], \dots\}$ where additions are done modulo n. The lookup follows the same rule, the array is searched from $T[h(x)]$ until either x is located, or a free cell is encountered, in which case x is not in the dictionary. Deletions cannot simply remove an item from the array, because this may invalidate the correctness of the lookup operation. Deletions need to compact the remainder if the run starting form the removed item. The details of how to do that are straight forward and omitted here.

The scheme is popular mainly due to the simplicity of the algorithm, but it turns out that it is very fast in practice. From a theoretical standpoint the running time clearly depends on the lengths of the 'runs' of occupied cells in T, these runs may clearly be larger than 2, thus the number of probes needed for a search may be higher than in cuckoo hashing. Further, it would be longer than the *uniform probing* scheme, where instead of scanning the array linearly, the order of scan is determined by some random permutation indexed by the query. In practice however there are two properties of computer architecture that play to Linear Probing's advantage. The first is memory locality. Since the probes of the scheme are consecutive elements in an array, the first probe already brings into the memory cache the remaining probe and thus if the run is not very long, the cost in time is effectively closer to that of one probe. Second, even if the run happens to be long, modern CPU's would often identify a linear memory scan and *pre-fetch* the next probe into the cache, again effectively reducing the running time.

Note that a cell $T[i]$ is empty iff for every k the number of items mapped to locations $k \dots i$ is less than $i - k$. It also implies that the set of occupied cells depends only upon the set of items that is inserted and is not a function of the order of insertion. We call a set of consecutive occupied cells a *run*. Let $\ell_1, \ell_2, \dots$ be the lengths of the runs, so that $\sum \ell_i = m$.

Lemma 5.13. *The expected number of probes in an unsuccessful search, assuming the searched key is mapped to a random location in the table is at most* $1 + \frac{1}{n} \sum \frac{\ell_i(\ell_i+1)}{2}$

Proof. Proof is done by linearity of expectation. With probability $1 - \frac{m}{n}$ the query is mapped to an empty slot and the search takes a single probe. With probability ℓ_i/n the query is mapped to the i'th run. Conditioned on this the number of probes is the length of the remaining run which is $(\ell_i + 1)/2$, plus one more probe at the empty cell to its right. So the total expected

number of probes is $\frac{\ell_i+1}{2}+1$ and summing up the expectation is

$$1-\frac{m}{n}+\frac{1}{n}\sum\left(\frac{\ell_i(\ell_i+1)}{2}+1\right)\leq 1+\frac{1}{n}\sum\frac{\ell_i(\ell_i+1)}{2}$$

□

Recall that $\alpha := m/n$ denotes the load of the table.

Theorem 5.14 (Knuth)**.** *Assuming the hash function is fully random, the expected time to insert an item, which is also the expected time for an unsuccessful lookup operation is*

$$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$$

The expected time for the successful lookup of a randomly selected item from the table is

$$\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$$

We show a different analysis for linear probing, originally in [85], and then simplified and a bit extended in [92]. The main advantage of this analysis is that it removes the random hashing assumption and replaces it with a 5−wise independence family. In fact, even 4−wise independence suffices for an amortized bound. This provides an explicit hash function family in the spirit of the classical works of Carter and Wegman [22], where each hash function could be indexed using $O(\log n)$ bits and could be evaluated in $O(1)$ time in the RAM word model. For sake of readability we will be loose with the dependency on α.

Following the exposition in [92] we make a few simplifying assumptions, first we assume that n the size of the table is a power of two. Second, we assume that $\alpha = m/n \leq 2/3$. So we will show a bound of $O(n)$ on the total running time without flushing out the dependency on α. On the other hand, the bound will hold for 4−wise independent families as well.

For an item x, $h(x)$ is the hash location in the table, x itself of course may be placed to the right of $h(x)$. In order to reason about the allocation of items into the memory arrays it is convenient to build a binary tree of depth $\log n$ whose leaves are the memory slots. We stress that this tree is not a part of the data structure but just a logical construct which is part of the proof. It is convenient to measure height from the leaves up, so a node of height i is the ancestor of 2^i leaves which are associated with consecutive

locations in the table. We call these leaves the interval of the node. Since the hash function is assumed to be uniform over the array, a node at height i is expected to have $\alpha 2^i \leq \frac{2}{3}2^i$ items hashed to its interval. We say the node is *crowded* if more than $\frac{3}{4}2^i$ are hashed to its interval. Note that the actual location in which the items are finally placed may be outside that interval.

Lemma 5.15. *Say after placing n items there is a run of length $\ell > 2^i$, $i \geq 3$. Then there is a* crowded *node at height $i-2$ with an interval whose last memory location is part of the run.*

Proof. Note that the interval of a node at height $i-2$ is of size 2^{i-2} so there are at least 4 nodes of height $i-2$ with their last node in the run, in fact for 3 of those nodes their entire interval is contained in the run, only the first of these nodes may only intersect the run at the suffix of its interval. We call these nodes $v_1, \ldots, v_4$. Assume for contradiction they are all not crowded. We count the total number of items placed in the run. Just before the run, there is an empty cell. Thus, there are no items that are placed in the run from the left of v_1, and v_1 contributes at most $(3/4)2^{i-2}$ items to the run. Since v_2, v_3, v_4 are also not crowded, each of them has at most $(3/4)2^{i-2}$ items mapped, so each can absorb at least $(1/4)2^{i-2}$ additional items. It must be therefore that the run ends at or before the interval of v_4 and its length is less than $4 \cdot 2^{i-2} = 2^i$ contradicting the assumption on the length of the fun. □

The total insertion time of n items is $O(\sum \ell_i^2)$. The runs where $i \leq 2$ can contribute at most $4n$. We have just seen that a longer run has a crowded ancestor at height 2^{i-2} whose last leaf is part of the run, so a crowded node is associated with at most one run. We have therefore

$$\mathbb{E}\left[\sum \ell_i^2\right] \leq 4n + O\left(\sum_v 2^{2\cdot \text{height}(v)} \Pr[v \text{ is crouded}]\right)$$

There are $n/2^i$ nodes at height i, so this is

$$O\left(\sum_{i=0}^{\log n} \frac{n}{2^i} 2^{2i} \Pr[\text{node of height } i \text{ is crouded}]\right)$$
$$= n \cdot O\left(\sum_{i=0}^{\log n} 2^i \Pr[v \text{ node of height } i \text{ is crouded}]\right) \tag{24}$$

The key for the analysis therefore is an effective bound on the probability a node is crowded, and in fact, any hash function with strong enough bound

would suffice. Denote the probability a node of height i is crowded by γ_i. As a first example consider a fully random function. In this case the load of v is distributed $B(m, 2^i/n)$. Cherrnoff's bound implies that there is some constant c so that $\gamma_i \leq \exp(-c2^i)$ which means that (24) is bounded by $O(n)$.

5.4.1 Five-wise independent hash functions

The key observation is that 24 could be bounded by $O(n)$ even when the hash function is not fully random, but rather drawn from a family of constant independence.

Definition 10. *Let H be a family of hash functions from $\mathcal{U}$ to $\mathcal{V}$. We say H is k-wise independent if for every distinct $x_1, \ldots, x_k \in \mathcal{U}$, when h is a function uniformly sampled from H, the variables $h(x_1), \ldots, h(x_k)$ are independent.*

Let $y_1, \ldots, y_n$ be a set of $k-$wise independent variables, let $Y = \sum y_i$ and $\mu = \mathbb{E}[Y]$. By the k independence it follows that $\mathbb{E}[(Y-\mu)^k] = O(\mu^{k/2})$. We use this to compute γ_i which is the probability a node of height i is crowded. Given a node at height i, let y_j be the probability the j'th item was hashed to the node's interval and $Y = \sum y_j$. Recall that $\mu = \mathbb{E}[Y] \leq \frac{2/3}{2}^i$

$$\begin{aligned}\gamma_i = \Pr[Y \geq \frac{3}{4}2^i] &\leq \Pr[|Y-\mu| \geq \frac{3}{24}\mu] \\ &\leq \Pr\left[(|Y-\mu|)^k \geq (\frac{3}{24}\mu)^k\right] \\ &\leq O(2^{-ik/2})\end{aligned}$$

where the last inequality is an application of Markov's inequality and the bound on the k'th moment of Y. Plugging this in 24 implies that the expected time for n insertions is $O(n)$ whenever $k \geq 4$, and that it is $O(n \log n)$ even when H is only pair-wise independent.

A slightly sharper bound was shown in [85]:

Theorem 5.16. *Let H be a 5-wise independent from $\mathcal{U} \to [n]$. When linear probing is used with a hash function chosen uniformly at random from H, the expected total number of probes made by a sequence of n insertions into an empty table is less than*

$$n \cdot \left(1 + O\left(\frac{\alpha}{(1-\alpha)^2}\right)\right)$$

Next we consider lookups. Consider an item x and its hash location $h(x)$. If $h(x)$ is in a run of length ℓ then the lookup time is $O(\ell)$. Now, say $2^i \leq \ell \leq 2^{i+2}$. We know already that at least one of the first 4 nodes at level $i-2$ whose intervals intersect the run are crowded. The hash location $h(x)$ could be at the beginning of the run or the end of it, so we conclude that *conditioned on* $h(x)$ and on $h(x)$ being part of a run of length ℓ, it must hold that at least one of 12 nodes at height $i-2$ are crowded. These 12 nodes are those that correspond to all possible locations of $h(x)$ within the run. If $i \leq 2$ lookup time is at most 3, so

$$\mathbb{E}[\ell \mid h(x)] \leq 3 + 12 \sum_{i \geq 2} 2^{i+1} \cdot \gamma'_{i-2} \tag{25}$$

where γ'_i as the probability a node of height i is crowded *conditioned* on $h(x)$. The conditioning on $h(x)$ takes one degree of freedom away from the hash function, the remaining values are $k-1$ independent. In other words:

$$\gamma'_i \leq \gamma_{i-1} = O(2^{-i(k-1)/2})$$

Plugging back in (25) we have that the expected time for a lookup is $O(\sqrt{n})$ when $k = 2$, $O(\log n)$ when $k = 4$ and remarkably $O(1)$ when $k \geq 5$.

5.5 Explicit hash functions

Section 5.4.1 is an example where the assumption the data structure uses a fully random hash function is removed. This assumption is one of the main discrepancies between the theoretical model and practice where the space it takes to represent the function and the time it takes to compute it must be taken into account. As such, specifying an explicit hash function family had been a prime goal of data-structure research from its beginning. A full review of this line of research is beyond the scope of this manuscript and deserves a separate survey. Below is a brief presentation of the main papers that may serve as a starting point for the interested reader.

The starting point is likely the seminal work of Carter and Wegman [22], where $k-$wise independent functions were introduced. The notion of limited independence turned out to be very useful, some dictionaries maintain their guarantees even when the hash function is drawn from such a family with $k = O(1)$, including for instance [49], [35] and [18]. As the level of independence k increases more applications could be found. Simple chaining could be implemented with $k = O(\log n)$. It was observed in [8] that the proof of 'vanila' cuckoo hashing in Section 5.1 (the $(2, 1)$ scheme) measures the probability of occurrence of structures of size $O(s \log n)$. A careful

observation reveals that setting $k = O(s \log n)$ suffices for essentially the same analysis to go through. If k is allowed to increase to $k = \log^{O(1)}(n)$ then more applications come under the fold via a more generic proof technique. The technique is based on a result by Braverman [16] that showed that AC_0 circuits of at most quasi-polynomial size cannot distinguish (with high probability) between the case their input is truly random to the case the input is drawn from a k-independent distribution where $k = \log^{O(1)} n$. AC_0 are the family of boolean circuits of constant depth and unlimited fan in. It was observed in [6] that Braverman's result implies that $k-$wise independent functions could replace truly random hash functions whenever the failure of the data structure under the full randomness assumption implies with high probability the occurrence of some'bad' event, and further that the bad event could be identified by an AC_0 circuit of quasi-polynomial size. They demonstrated this technique by applying if for the de-amortized cuckoo hashing (see Section 5.2). In this case the bad event is the existence of some dense sub-graph of the cuckoo-graph. The key observation is that the dense sub graphs are of logarithmic size, and therefore there are a quasy-polynomial number of such sub graphs, so their existence could be checked in parallel by a constant depth and quasi-polynomial size circuit.

There are many ways to construct (almost) k-wise independent functions c.f. [22], [104], [4], [3], [1], [79] [31]. These constructions arise in many different contexts such as error correcting codes and epsilon biases spaces. See for instance [5]. A canonical one is based on evaluating a degree $k-1$ polynomial. Storing a degree $k-1$ polynomial entails storing k coefficients, so it takes $O(k)$ space (in the word RAM model). Computing the function is done by evaluating the polynomial on the key, which takes $O(k)$ time. Thus, when used for many of the constructions for dictionaries that we have seen, if k is super constant, the time to compute the hash function dominates the running time of the insertion or lookup algorithms. It is therefore crucial to use a hash function that could be evaluated in $O(1)$ time. Indeed, this problem did not go unnoticed, an intriguing line of research is aimed at finding (almost) $k-$wise independent families for large k where the evaluation time is $O(1)$, at the cost of using much more space. The first example of such a family was given by Siegel in [97]. It shows a family of functions that are almost n^δ independent using $n^{\delta'}$ space where $\delta, \delta' < 1$. Siegel's construction is optimal asymptotically but the constants are enormous. A much more efficient construction is given in [25] and is based on tabulation.

It is worth noting that even for $k = 5$, evaluating a polynomial hash function would in practice be more expensive than a lookup in the table,

the reason being in practice multiplication is fairly expensive. So it is worth while to find more efficient constructions for this case. A construction based on a simple version of tabulated hashing was shown in [102] and for larger k in [62]. Tabulated hashing essentially requires just reading and xoring a few locations in the memory indexed by the key itself.

A slightly different notion of limited independence is that of *simulated randomness*. In this setting for every set of n items from the universe $x_1, \ldots x_n$, the values $h(x_1), \ldots, h(x_n)$ are *fully random* with probability $1 - 1/poly(n)$. With probability $1/poly(n)$ this particular set of keys are not good for the sampled hash function and so these values may be correlated in an arbitrary way. Clearly a construction with this property would require linear space. Surprisingly in [84] and [38] it was shown that families with this property could be evaluated in constant time. Simulated randomness is useful not only when the cost of linear space is acceptable. A standard 'trick' is to partition the set of items to a sub linear of sets with $o(n)$ in each set, and then using the same hash function across all sets [36].

While the notion of limited independence is appealing in its generality, specialized hash functions turn to be very efficient for specific data structures. Simple tabulated hashing and some of its variants [88], [91], [28] are only 3−wise independent yet could be used for linear probing, the 2-coice scheme and a simple chained dictionary. Other examples include [8] for cuckoo hashing and [106] for d-Left hashing.

Another goal is to reduce space. For instance, for simple chaining (the one choice scheme), fully random functions could be replaced by $O(\log n)$-wise independent functions. This takes $O(\log n)$ space. A hash function family with similar performance and $O(\log logn)$ space was shown in [24]. In [94] it was shown that this family could be used for some versions of cuckoo hashing as well. In [70] it was shown that these functions could also be evaluated in $O(\log logn)$ time. An intriguing open question [2] asks whether there is a family of hash functions in which a hash function could be represented in $O(1)$ space, and yet could be used in the one choice scheme with essentially the same guarantees as a fully random function.

It was observed in [94] that whenever the 'misbehavior' of a fully random hash function can be characterized with high probability by a bad event, and this event could be detected by an algorithm with memory s, the fully random hash function could be replaced by one based on a pseudo-random generators for space bounded computation. This observation is similar to the one made previously regarding AC_0 circuits. They use the generators of [83] and [69] to build a hash function with $O(\sqrt{\log n})$ space for the one choice scheme.

A surprising fact is that in practice and in experiments it is often the case that very simple hash functions, say a simple linear function, are just as good as the pseudo-random functions with the provable guarantees. A possible explanation for this phenomena was given in [76]. They show that if the *keys* themselves have sufficient min-entropy than even very simple hash functions could be used. The tool they use is the leftover hash lemma which states that pairwise independent functions are in fact some sort of weak extractor. The main observation is that the leftover hash lemma implies that if the keys are drawn from some weak distribution, the output of a pairwise independent function is close enough to that of a fully random function so that the guarantees of full randomness still hold. The negative results of [37] demonstrate that in common cases where the keys have some structure and little entropy, pairwise independent can behave poorly, so the results in [76] should not be taken as a generic justification for using such weak functions.

References

[1] Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.

[2] Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos. Linear hash functions. *J. ACM*, 46(5):667–683, September 1999.

[3] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.

[4] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Addendum to "simple construction of almost k-wise independent random variables". *Random Struct. Algorithms*, 4(1):119–120, 1993.

[5] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 1992.

[6] Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 107–118, 2009.

[7] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 787–796, 2010.

[8] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014.

[9] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, September 1999.

[10] Tugkan Batu, Petra Berenbrink, and Colin Cooper. Chains-into-bins processes. *J. Discrete Algorithms*, 14:21–28, 2012.

[11] Petra Berenbrink, André Brinkmann, Tom Friedetzky, and Lars Nagel. Balls into non-uniform bins. *J. Parallel Distrib. Comput.*, 74(2):2065–2076, 2014.

[12] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006.

[13] Petra Berenbrink, Tom Friedetzky, Zengjian Hu, and Russell A. Martin. On weighted balls-into-bins games. *Theor. Comput. Sci.*, 409(3):511–520, 2008.

[14] Petra Berenbrink, Kamyar Khodamoradi, Thomas Sauerwald, and Alexandre Stauffer. Balls-into-bins with nearly optimal load distribution. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pages 326–335, 2013.

[15] Paul Bogdan, Thomas Sauerwald, Alexandre Stauffer, and He Sun. Balls into bins via local search. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 16–34, 2013.

[16] Mark Braverman. Polylogarithmic independence fools ac0 circuits. *J. ACM*, 57(5):28:1–28:10, June 2008.

[17] Karl Bringmann, Thomas Sauerwald, Alexandre Stauffer, and He Sun. Balls into bins via local search: cover time and maximum load. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 187–198, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[18] Andrei Z. Broder and Anna R. Karlin. Multilevel adaptive hashing. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 43–53, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[19] Andrei Z. Broder and Michael Mitzenmacher. Using multiple hash functions to improve IP lookups. In *Proceedings IEEE INFOCOM 2001, The Conference on Computer Communications, Twentieth Annual Joint Conference of the IEEE Computer and Communications*

Societies, Twenty years into the communications odyssey, Anchorage, Alaska, USA, April 22-26, 2001, pages 1454–1463, 2001.

[20] John Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple load balancing for distributed hash tables. In *IPTPS*, pages 80–87, 2002.

[21] Julie Anne Cain, Peter Sanders, and Nicholas C. Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 469–476, 2007.

[22] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.

[23] Cassandra. Apache. http://cassandra.apache.org/.

[24] L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. *SIAM J. Comput.*, 42(3):1030–1050, 2013.

[25] Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. From independence to expansion and back again. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 813–820, 2015.

[26] Artur Czumaj and Volker Stemann. Randomized allocation processes. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 194–203, 1997.

[27] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over k-partitions. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1292–1310, 2015.

[28] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1631–1642, 2016.

[29] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[30] Luc Devroye and Pat Morin. Cuckoo hashing: Further analysis. *Information Processing Letters*, 86:215–219, 2003.

[31] Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22-24, 1996, Proceedings*, pages 569–580, 1996.

[32] Martin Dietzfelbinger, Joseph Gil, Yossi Matias, and Nicholas Pippenger. Polynomial hash functions are reliable (extended abstract). In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, ICALP '92, pages 235–246, London, UK, UK, 1992. Springer-Verlag.

[33] Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight thresholds for cuckoo hashing via XORSAT. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, pages 213–225, 2010.

[34] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, ICALP '90, pages 6–19, London, UK, UK, 1990. Springer-Verlag.

[35] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994.

[36] Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ICALP '09, pages 354–365, Berlin, Heidelberg, 2009. Springer-Verlag.

[37] Martin Dietzfelbinger and Ulf Schellbach. On risks of using cuckoo hashing with simple universal hash classes. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 795–804, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[38] Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 629–638, 2003.

[39] Martn Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007.

[40] Gregory Dresden and Du Zhaohui. A simplified binet formula for k-generalized fibonacci numbers. *Journal of Integer Sequences*, 17, 2014.

[41] Michael Drmota and Reinhard Kutzelnigg. A precise analysis of cuckoo hashing. *ACM Trans. Algorithms*, 8(2):11, 2012.

[42] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

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

[44] Daniel Fernholz and Vijaya Ramachandran. The k-orientability thresholds for $G_{n,\ p}$. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 459–468, 2007.

[45] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005.

[46] Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The multiple-orientability thresholds for random hypergraphs. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1222–1236, 2011.

[47] Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp load thresholds for cuckoo hashing. *Random Struct. Algorithms*, 41(3):306–333, 2012.

[48] Nikolaos Fountoulakis, Konstantinos Panagiotou, and Angelika Steger. On the insertion time of cuckoo hashing. *SIAM J. Comput.*, 42(6):2156–2181, 2013.

[49] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, June 1984.

[50] Alan Frieze, Páll Melsted, and Michael Mitzenmacher. An analysis of random-walk cuckoo hashing. *SIAM J. Comput.*, 40(2):291–308, March 2011.

[51] Alan M. Frieze and Tony Johansson. On the insertion time of random walk cuckoo hashing. *CoRR*, abs/1602.04652, 2016.

[52] Alan M. Frieze and Páll Melsted. Maximum matchings in random bipartite graphs and the space utilization of cuckoo hash tables. *Random Struct. Algorithms*, 41(3):334–364, 2012.

[53] Pu Gao and Nicholas C. Wormald. Orientability thresholds for random hypergraphs. *Combinatorics, Probability & Computing*, 24(5):774–824, 2015.

[54] P. Brighten Godfrey. Balls and bins with structure: Balanced allocations on hypergraphs. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 511–517, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[55] Michael Goodrich, Evgenios Kornaropoulos, Michael Mitzenmacher, and Roberto Tamassia. More practical and secure history-independent hash tables. In *European symposium on Research in Computer Security*, 2016.

[56] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[57] Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient pram simulation on a distributed memory machine. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 318–326, New York, NY, USA, 1992. ACM.

[58] Krishnaram Kenthapadi and Rina Panigrahy. Balanced allocation on graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 434–443, 2006.

[59] Megha Khosla. Balls into bins made faster. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 601–612, 2013.

[60] Adam Kirsch and Michael Mitzenmacher. Using a queue to de-amortized cuckoo hashing in hardware. In *the Forty Fifth Annual Allerton Conference on Communication, Control, and Computing*, 2007.

[61] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, December 2009.

[62] Toryn Qwyllyn Klassen and Philipp Woelfel. Independence of tabulation-based hash classes. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium, Arequipa, Peru, April 16-20, 2012. Proceedings*, pages 506–517, 2012.

[63] Donald Knuth. Notes on open adressing. *http://citeseer.ist.psu.edu/knuth63notes.html*, 1963.

[64] Reinhard Kutzelnigg. A further analysis of cuckoo hashing with a stash and random graphs of excess r. *Discrete Mathematics & Theoretical Computer Science*, 12(3):81–102, 2010.

[65] Eric Lehman and Rina Panigrahy. 3.5-way cuckoo hashing for the price of 2-and-a-bit. In Amos Fiat and Peter Sanders, editors, *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 671–681. Springer, 2009.

[66] Marc Lelarge. A new approach to the orientation of random hypergraphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Sym-*

posium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012, pages 251–264, 2012.

[67] Daniel Lemire. The universality of iterated hashing over variable-length strings. *Discrete Applied Mathematics*, 160(4-5):604–617, 2012.

[68] Christoph Lenzen and Roger Wattenhofer. Tight bounds for parallel randomized load balancing: Extended abstract. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 11–20, New York, NY, USA, 2011. ACM.

[69] Chi-Jen Lu. Improved pseudorandom generators for combinatorial rectangles. *Combinatorica*, 22(3):417–434, 2002.

[70] Raghu Meka, Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Fast pseudorandomness for independence and load balancing - (extended abstract). In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 859–870, 2014.

[71] Daniele Micciancio. Oblivious data structures: Applications to cryptography. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 456–464, New York, NY, USA, 1997. ACM.

[72] Michael Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, Harvard University, 1991.

[73] Michael Mitzenmacher. Balanced allocations and double hashing. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, Prague, Czech Republic - June 23 - 25, 2014*, pages 331–342, 2014.

[74] Michael Mitzenmacher, Andra W. Richa, and Ramesh Sitaraman. The power of two random choices: A survey of techniques and results. In *in Handbook of Randomized Computing*, pages 255–312. Kluwer, 2000.

[75] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

[76] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *Proceedings of the*

Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '08, pages 746–755, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[77] Michael Mitzenmacher and Berhold Vöcking. The asymptotics of selecting the shortest of two, improved. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, pages 326–327, 1999.

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

[79] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[80] Moni Naor, Gil Segev, and Udi Wieder. History-independent cuckoo hashing. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, pages 631–642, 2008.

[81] Moni Naor and Vanessa Teague. Anti-persistence: History independent data structures. In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 492–501, New York, NY, USA, 2001. ACM.

[82] Moni Naor and Udi Wieder. Novel architectures for p2p applications: The continuous-discrete approach. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, pages 50–59, New York, NY, USA, 2003. ACM.

[83] Noam Nisan and David Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, February 1996.

[84] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[85] Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with 5-wise independence. *SIAM Review*, 53(3):547–558, 2011.

[86] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, May 2004.

[87] Rina Panigrahy. Efficient hashing with lookups in two memory accesses. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 830–839, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[88] Mihai Patrascu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14, 2012.

[89] Yuval Peres, Kunal Talwar, and Udi Wieder. Graphical balanced allocations and the (1 + beta)-choice process. *Random Structures and Algorithms*, 2014.

[90] W. W. Peterson. Addressing for random-access storage. *IBM J. Res. Dev.*, 1(2):130–146, April 1957.

[91] Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 209–228, Philadelphia, PA, USA, 2013. Society for Industrial and Applied Mathematics.

[92] Mihai Pătraşcu and Mikkel Thorup. On the k-independence required by linear probing and minwise independence. *ACM Trans. Algorithms*, 12(1):8:1–8:27, November 2015.

[93] Martin Raab and Angelika Steger. balls into bins a simple and tight analysis. In Michael Luby, JosD.P. Rolim, and Maria Serna, editors, *Randomization and Approximation Techniques in Computer Science*, volume 1518 of *Lecture Notes in Computer Science*, pages 159–170. Springer Berlin Heidelberg, 1998.

[94] Omer Reingold, Ron D. Rothblum, and Udi Wieder. Pseudorandom graphs in data structures. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 943–954, 2014.

[95] A. Wayne Roberts and Dale E. Varberg. *Convex Functions*. Academic Press Inc., New York, NY, USA, 1st edition, 1973.

[96] Peter Sanders, Sebastian Egner, and Jan Korst. Fast concurrent access to parallel disks. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 849–858, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[97] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, 2004.

[98] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[99] Kunal Talwar and Udi Wieder. Balanced allocations: the weighted case. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 256–265, 2007.

[100] Kunal Talwar and Udi Wieder. Balanced allocations: A simple proof for the heavily loaded case. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, volume 8572 of *Lecture Notes in Computer Science*, pages 979–990. Springer Berlin Heidelberg, 2014.

[101] Mikkel Thorup. String hashing for linear probing. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 655–664, 2009.

[102] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM J. Comput.*, 41(2):293–331, 2012.

[103] Berthold Vöcking. How asymmetry helps load balancing. *J. ACM*, 50(4):568–589, July 2003.

[104] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.

[105] Udi Wieder. Balanced allocations with heterogenous bins. In *Proceedings of the Nineteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '07, pages 188–193, New York, NY, USA, 2007. ACM.

[106] Philipp Woelfel. Asymmetric balanced allocation with simple hash functions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 424–433, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.