Two-way Hashing with
Separate Chaining and Linear Probing

Ebrahim Malalla

School of Computer Science
McGill University, Montreal
March 2004

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements of the degree of Doctor of Philosophy

To my heros

Abstract

Two-way chaining is a novel hashing scheme that uses two independent truly uniform hash functions f and g to insert m keys into a hash table with n chains, where each key x is inserted into the shortest chain among the chains $f(x)$ and $g(x)$, breaking ties randomly. It is known [13, 18] that the worst-case search time of two-way chaining is $\log_2 \log n + m/n + O(1)$, asymptotically almost surely. In this thesis, we study the two-way chaining paradigm under different assumptions.

First, we generalize the result to nonuniform hash functions. We analyze two-way chaining in the fixed density model where the two independent hash functions behave according to two densities defined on the unit interval. When $m = \Omega(n)$, we prove that asymptotically almost surely, the worst-case search time is at least $\log_2 \log n - O(1)$. If, in addition, the densities are bounded, then it is at most $\log_2 \log n + O(m/n)$.

Secondly, we consider the off-line version of two-way chaining where all the hashing values available for the m keys are known in advance. For constant $k \in \mathbb{N}$, we show that there is a threshold c_k such that if $m \leq c_k n$, then one can assign the keys to the chains so that the maximum search time is at most $2k$, asymptotically almost surely. We tightly estimate c_k, and prove that it is, in fact, asymptotic to k. Algorithms for finding such assignments are also given.

Thirdly, we utilize the two-way chaining paradigm to design efficient open addressing hashing schemes. We study two-way linear probing algorithms. These are algorithms that employ two independent linear probe sequences to hash the keys. We prove an $\Omega(\log \log n)$ universal lower bound on the worst-case search time of any two-way linear probing algorithm, where n is the hash table size. We show, however, that some simple two-way linear probing algorithms, unexpectedly, have implausible worst-case performances. Subsequently, we present several efficient two-way linear probing algorithms whose performance matches the lower bound. Simulations back up the theoretical results.

Résumé

L'enchaînement à deux choix est une méthode de hachage qui emploie deux fonctions uniformes indépendantes f et g pour insérer m clefs dans une table avec n chaînes, où chaque clef x est insérée dans la chaîne la plus courte parmi les chaînes $f(x)$ et $g(x)$. On sait que le maximum des temps de recherche est $\log_2 \log n + m/n + O(1)$, asymptotiquement presque sûrement. Dans cette thèse, nous étudions le paradigme d'enchaînement à deux choix dans différents contextes.

D'abord, nous généralisons le résultat aux fonctions de hachage non-uniformes. Nous analysons l'enchaînement à deux choix dans le modèle de densité où les deux fonctions de hachage se comportent selon deux densités définies sur l'intervalle d'unité. Quand $m = \Omega(n)$, nous prouvent cela asymptotiquement presque sûrement, le temps maximal de recherche est au moins $\log_2 \log n - O(1)$. Si, en outre, les densités sont bornées, alors il est tout au plus $\log_2 \log n + O(m/n)$.

En second lieu, nous considérons la version off-line où toutes les valeurs de hachage pour les m clefs sont connues à l'avance. Pour la constante $k \in \mathbb{N}$, nous prouvons qu'il y a un seuil c_k tel que si $m \leq c_k n$, on peut assigner les clefs aux chaînes de sorte que le temps maximum de recherche soit tout au plus $2k$, asymptotiquement presque sûrement. Nous estimons c_k, et montrons qu'il est, en fait, asymptotique à k. Des algorithmes efficaces sont donnés.

Troisièmement, nous utilisons le paradigme d'enchaînement à deux choix pour concevoir des algorithmes de hachage du type "open addressing". Nous proposons des algorithmes linéaires à deux choix. Ce sont des algorithmes qui utilisent deux recherches linéaires à partir de deux fonctions de hachage indépendantes. Nous prouvons une limite inférieure en universelle de $\Omega(\log \log n)$ pour le temps maximum de recherche de n'importe quel algorithme de hachage linéaire à deux choix, où n est la taille de la table. Nous montrons, cependant, que quelques algorithmes de hachage linéaires à deux choix simples, inopinément, avons des exécutions des cas les pires décevantes. Nous présentons deux algorithmes de hachage linéaires à deux choix efficaces dont la performance est optimale en n. Des résultats de simulation confirment les propriétés théoriques.

Acknowledgements

And surely your Lord is full of bounty for mankind, but most of them do not give thanks. And verily your Lord knows what their hearts hide, and what they reveal.

THE NOBLE QUR'AN, (27: 73–74)

Writing this acknowledgement is one of the joyful moments that I have dreamed of, countless times, during the last years. The praise is for Allah, the most merciful, the most compassionate, for blessing me beyond deserving with the support of many people who have had a profound impact on my life. A word of thank you to these people is certainly not enough.

Referring to the humongous difficulty of teaching me, and the heavy responsibility ahead of him, my supervisor once said to me: "I used to have only two daughters, and now, I have a new son!" From the time of accepting to be my supervisor to this wonderful moment, Luc Devroye has never stopped guiding me in the right direction. His continuous and monotonically increasing help goes beyond definition. His valuable advice, suggestions, and corrections improved this work dramatically. Indeed, every beautiful idea in this thesis was originally conceived in his mind. I am deeply indebted for his warm kindness, infinite patience, and absolute generosity. It is my pleasure to express my sincere gratitude to him. Luc, it is an honor to be your student.

Appreciations must be expressed to the friendly colleagues and professors of the School of Computer Science for the warm congenial atmosphere. Particular thanks are due to Ketan Dalal for his help in improving the simulation programs.

I would like also to take this opportunity to declare my indebtedness to the incredible people who played a pivotal role in my life. I am most indebted to the man who showed me the beauty of pure thinking, to my favorite mathematician, Roshdi Khalil: you are always an inspiration. To the friends who encouraged and supported me when I needed them. Every single member of my family, to whom I dedicate this thesis, deserves a thank you for believing in me. My deepest gratitude goes out to my dear brother Abu Hussain, and my beloved sister Om Hassan. At the top are my mom and dad who always believe that I can fly!

Montreal, March 2004 *E. M.*

Contents

Introduction

Since its invention in the middle of the last century, hashing has never been more appealing than today. Its presence in many branches of computer science has motivated many researchers to find new creative ways for improving its performance. While the average performance of hashing is clearly a crucial factor in practice, its worst-case performance cannot be ignored. The last decade has witnessed the birth of new hashing schemes that advance the worst-case performance of hashing to a plausible level. This thesis is a humble step on the same road focusing only on the worst-case performance of hashing.

A classical hash table implementation [103, 80, 169] uses one hash function f to insert m distinct input keys that come from a finite universe set of keys $\mathcal{U}$ into a table of size n. The hash table is a one-dimensional array with n cells which we denote, throughout, by the set $\mathcal{T} := \{0, \ldots, n-1\}$. The ratio $\alpha := m/n$ is called the load factor of the hash table. In an ideal situation, the hash function f would be *perfect*, that is, an injective function. A key x, then, is hashed to the cell $f(x)$. Throughout, we define the *insertion* and *search times* to be the number of probes (table accesses) needed to insert or locate a key, respectively, plus the time required to compute the hashing addresses. For simplicity, we ignore, throughout, the evaluation time of the hash functions.

Many techniques have been developed to derive perfect hash functions. However, all of them, understandably, are *off-line* techniques, that is, they require prior knowledge of the input keys. Usually, they work only with *static* hash tables which means that the keys are not allowed to be updated. Alternatively, the hash function f can be chosen uniformly at random from the set of all possible functions that map $\mathcal{U}$ into $\mathcal{T}$. In this case, we say that the hash function f is *truly uniform* as its hashing values are independent and uniformly distributed over the hash table. The function also is independent of the input keys. However, the birthday paradox [66] reveals that when $m = \omega(\sqrt{n})$, then with high probability (that is, with probability goes to one as n goes to infinity), there are two keys that will be mapped to the same cell. That is, a *collision* will occur. Several collision resolutions have been devised. Among these are *separate chaining* and *open addressing*. For a historical background of these methods and others see [103, 121].

Hashing with Separate Chaining

Collisions can be resolved by allowing each cell in the hash table to have a separate linked list or chain. Keys that hash to a certain cell are inserted into the chain pointed to by the cell.

Classical Uniform Chaining

Classically, a truly-uniform hash function f is used to insert m keys sequentially into a table with n chains, where each key x is appended to the chain $f(x)$. The insertion time is constant, and the search time of any key x is at most the length of the chain $f(x)$, where the length of a chain is defined to be the number of keys the chain contains. The average chain length α [103, 80, 169] is bounded if α can be kept bounded. This is not the case with the maximum search time which is proportional to the longest chain length. Gonnet [79] showed that for constant load factor, the maximum chain length is asymptotic to $\log n / \log\log n$, in probability. This fact has been proved earlier in terms of balls and bins, see [99, 105]. Other proofs appeared more recently in [130, 153].

Two-way Uniform Chaining

Azar, Broder, Karlin and Upfal [13] suggested a new hashing scheme called *two-way chaining*. Two independent and truly-uniform hash functions f and g are used to hash the keys sequentially. Each key x is inserted into the shortest chain among the chains $f(x)$ and $g(x)$, breaking ties randomly. Assuming that we save with each chain its length, the insertion time is still constant. To search for any key x, we check both chains $f(x)$ and $g(x)$. Thus, the average search time is not more than twice the average search time of classical uniform hashing with chaining explained above. The maximum search time is at most twice the length of the longest chain. However, Azar et al. [13] proved that when the load factor α is constant, the longest chain length decreases dramatically to $\log_2 \log n \pm \Theta(1)$, with high probability.

The two-way chaining paradigm has provoked an avalanche of research [18, 24, 39, 114, 134, 170, 177]. The hashing scheme has several advantages over other proposed methods that lead to plausible worst-case performance like the ones in [74, 23, 50]. It uses only two hash functions, it is easy to parallelize, it does not involve rehashing of data, and it is on-line and suitable for dynamic hashing.

Nonuniform Chaining

Truly uniform hash functions tend to distribute the keys evenly over the hash table; and hence, if $\mathcal{U}$ is an ordered set, these functions are, most likely, not order-preserving. Uniform order-preserving hash functions can be designed if the key statistics are known priori [155, 76]. If the order-preserving hash function is independent of the key distribution, the hashed values are typically nonuniformly distributed over the hash table, see, e.g., [82] and [41, p. 2]. Order-preserving functions are helpful for operations that require sorted or nearly sorted keys like range search and finding the k-nearest neighbors; see [42] for such applications. Lately, there has been growing interest in locally-sensitive hash functions [112, 91, 77, 25, 160]. These functions are sensitive to the similarity of the keys: they map keys that are similar to close chains. Evidently, the image of a locally-sensitive function also has a possibly nonuniform distribution. All of this underlines the importance of studying the performance of hashing schemes with nonuniform hash functions.

The worst case performance of classical chaining with nonuniform distributions was studied by Devroye [41]. He represented the hash table by the unit interval $[0, 1]$ partitioned into n equal-sized subintervals. The hashing locations of the keys, say $Y_1, \ldots, Y_m$, are assumed to be independent and have a common density function h over $[0, 1]$. The t-th key is hashed to the i-th chain, if Y_t belongs to the i-th subinterval. For constant load factor and bounded density h, he showed that the expected maximum chain length is still asymptotic to $\log n / \log \log n$. A tight upper bound is also given for unbounded densities. This leads us to study under which circumstances the bounds proved for two-way uniform chaining remain valid with nonuniform hash functions.

Off-line and Static Uniform Chaining

In the off-line version of two-way uniform chaining, the choices of hashing addresses available for all keys, where each key has two choices, are known in advance, before the insertion process starts. One can ask, then, if it is possible to assign each key to one of its two hashing addresses in a way that minimizes the length of the longest chain. Notice that the hashing choices are still independent and uniformly distributed as they are the images of truly uniform hash functions. The problem is useful for constructing efficient static hashing schemes, and giving insight into the competitive analysis of two-way chaining where the performance of the on-line version is compared to its off-line correspondent. Czumaj and Stemann [39] studied the problem, and proved that if $m \leq 1.67545943... \times n$, then with high probability there exists an assignment for the keys such that the maximum chain length is at most two. But can we improve the bound on m? What about higher maximum chain lengths? Let m_k, for $k \geq 2$, be the maximum m such that there is an assignment where the maximum chain length is at most k, with high probability? Czumaj and Stemann's analysis implies that $2m_k \geq k + \sqrt{k \log k} + \Omega(\log k)$, for k large enough. But can we do better? It is a question we shall address.

Thesis Contributions: Part I

The thesis is divided into two parts. The unifying theme of the thesis is the worst-case performance of two-way hashing methods using chaining and open addressing. The first part of the thesis is devoted to our contributions in two-way chaining.

Chapter 1: We give a new proof of the lower bound on the length of the longest chain produced by the on-line two-way uniform chaining algorithm, and we simplify the so called *witness tree method* (used in [170]) to prove the upper bound.

Chapter 2: We analyze the worst-case performance of on-line two-way chaining with independent nonuniform hash functions, f and g. Our analysis is based on the following *fixed density model*. We assume that the hash functions f and g map $\mathcal{U}$ to the unit interval $[0, 1]$ which is partitioned into n equal-sized subintervals where each subinterval represent a chain. All hashing locations are independent, and for each key x, the values $f(x)$ and $g(x)$ behave according to independent fixed densities

h_f and h_g, respectively, over $[0,1]$. We prove that when m keys are inserted into n chains using this model, where m/n is constant, the maximum chain length is at least $\log_2 \log n - \Theta(1)$, with high probability; and if the densities are bounded by some constants, then it is at most $\log_2 \log n + \Theta(1)$, with high probability. Upper bounds for unbounded densities with some conditions are also studied. Bounds for other cases such as the heavily- and lightly-loaded cases, or the dynamic case are also given.

Chapter 3: We extend the results in the literature for off-line two-way uniform chaining. We reduce the assignment problem to an orientation problem in a random graph with n vertices (chains) and m edges (keys). We show that there is an assignment for the keys where the maximum chain length is at most k, for $k \geq 2$, if and only if the random graph is *k-orientable*, that is, if there exists an orientation of the edges such that the maximum out-degree is at most k. The problem now is to estimate the maximum number of edges (keys) m_k such that the random graph is k-orientable, with high probability. We give another proof for a characterization by Frank and Gyárfás [73] that *any graph is k-orientable if and only if the number of edges of any subgraph is at most k times the number of its vertices.* We use this fact to approximate m_k for small k, and we show that for k large enough, $1 - 2^k \exp(-k + 1 + e^{-k/4}) < m_k/(kn) < 1 - \exp\left(-2k\left(1 - e^{-2k}\right)\right)$. Algorithms for finding a k-orientation are also presented.

Chapter 4: Finally, we discuss some of the speedups of two-way chaining, and the trade-offs between the number of hashing choices for each key, the search and insertion times, and the memory size.

Hashing with Open Addressing

Another method for resolving collisions is open addressing. The hash table does not have chains, and each cell can harbor at most one key. However, each key x has an infinite probe sequence $f_i(x) \in \mathcal{T}$, for $i \in \mathbb{N}$ which it follows sequentially until an empty cell is found where a key is inserted. The probe sequence is combined with a replacement strategy. During the insertion process, if a key x initiates the i-th probe and arrives at the cell $f_i(x)$ that is already occupied by another previously inserted key y, i.e., $f_i(x) = f_j(y)$, for some $j \in \mathbb{N}$, then a replacement strategy is used to resolve the collision. The strategy could be one of the following:

1. FIRST COME FIRST SERVED (FCFS) [147]: The key y is kept in its cell, and the key x is referred to the next cell $f_{i+1}(x)$.

2. LAST COME FIRST SERVED (LCFS) [151]: The key x is inserted into the cell $f_i(x)$, and the key y is pushed along to the next cell in its probe sequence, $f_{j+1}(y)$.

3. ROBIN HOOD [29, 28]: The key which travelled the furthest is inserted into the cell. That is, if $i > j$, then the key x is inserted into the cell $f_i(x)$, and the key y is pushed along to the next cell $f_{j+1}(y)$; otherwise, y is kept in its cell, and the key x tries its next cell $f_{i+1}(x)$.

There are many types of probe sequences, but the commonly used ones are:

1. RANDOM PROBING [136]: For every key x, the infinite sequence $f_i(x)$ is assumed to be independent and uniformly distributed over $\mathcal{T}$. That is, we require to have an infinite sequence f_i of truly uniform and independent hash functions. If for each key x, the first n probes of the sequence $f_i(x)$ are distinct, i.e., it is a random permutation, then it is called uniform probing [147].

2. LINEAR PROBING [147]: For every key x, the first probe $f_1(x)$ is assumed to be uniform on $\mathcal{T}$, and the next probes are defined by $f_{i+1}(x) = f_i(x) + 1 \mod n$, for $i = 1, \ldots, n$. So we only require f_1 to be a truly uniform hash function.

Random and uniform probings are, in some sense, the idealized models [164, 178], and their plausible performances are among the easiest to analyze; but obviously they are unrealistic. Linear probing is perhaps the simplest to implement, but it behaves poorly when the table is almost full.

Classical Open Addressing

In classical open addressing hashing, m keys are inserted, on-line and sequentially, into a table of size n by using only one probe sequence with a common replacement strategy. When we search for a key x, we have to follow the probe sequence $f_i(x)$ sequentially until the key is found or an empty cell in the case of unsuccessful search. The load factor $\alpha \in (0, 1)$ is assumed to be a constant. The asymptotic average-case performance has been extensively analyzed for different types of probe sequences [103, 80, 169]. The expected search times were proven to be constants, more or less, depending on α only. We focus, however, on the worst-case search time which is proportional to the length of the longest probe sequence over all keys (LLPS, for short).

Pittel [149] proved that in linear probing with FCFS policy, the LLPS needed to insert (or search for) any key is asymptotic to $(\alpha - 1 - \log \alpha)^{-1} \log n$, in probability. Gonnet [79] proved that with uniform probing and FCFS replacement strategy, the expected LLPS is asymptotic to $\log_{1/\alpha} n - \log_{1/\alpha} \log_{1/\alpha} n + O(1)$. However, Poblete and Munro [151, 152] showed that if random probing is combined with LCFS policy, then the expected LLPS is at most $(1 + o(1))\Gamma^{-1}(\alpha n) = O(\log n / \log \log n)$, where Γ is the gamma function.

On the other hand, the ROBIN HOOD strategy with random probing leads to a more striking performance. Celis [28] first proved that the expected LLPS is $O(\log n)$. However, Devroye, Morin and Viola [45] tightened the bounds and revealed that the LLPS is indeed $\log_2 \log n \pm \Theta(1)$, w.h.p., thus achieving double logarithmic worst-case insertion and search times for the first time in classical open addressing hashing. Unfortunately, one cannot ignore the unrealistic assumption in random probing about the availability of an infinite collection of independent and truly uniform hash functions. On the other side of the spectrum, it is known [147, 103] that the LLPS in linear probing, which is more realistic, is independent of the replacement strategy, because the insertion of any order of the keys results in the same set of occupied cells. This emphasizes the need for inventing nonclassical linear probing schemes.

Thesis Contributions: Part II

Our chief objective in the second part of this thesis is to design on-line linear probing schemes that achieve double logarithmic worst-case performance. This is done by exploiting the idea behind the two-way chaining paradigm. We promote the concept of *two-way linear probing*. These are hashing algorithms that initiate for each key two independent linear probe sequences with FCFS policy to find two empty cells where the key is inserted into one of them according to some strategy. For example, one of the trivial strategies inserts each key into the empty cell found by the shortest probe sequence. Another simple strategy inserts each key into the empty cell adjacent to the smallest cluster, where a cluster is an isolated set of consecutively occupied cells.

Chapter 5: We prove an $\Omega(\log \log n)$ universal lower bound on the performance of any strategy that uses two linear probe sequences, even if the starting points of these sequences are chosen according to arbitrary probability distributions. Furthermore, we demonstrate that not every two-way linear probing algorithm behaves nicely. We show, for instance, that when any of the above two strategies is used to construct a hash table with constant load factor, the maximum unsuccessful search time is $\Omega(\log n/\log\log n)$, with high probability.

Chapter 6: We introduce, subsequently, two on-line two-way linear probing algorithms that accomplish $\Theta(\log\log n)$ worst-case unsuccessful search time, with high probability. Simulation results that support the analysis of these algorithms are also presented. We study the performance of off-line two-way open addressing.

Say: bring your proof, if you are truthful.

The Noble Qur'an, (2: 111), (27: 64)

Chapter 0

Preliminaries

In this chapter we define some of the notations and recall some useful results from probability theory and analysis of algorithms.

0.1 Basic Notations

Throughout, we use $\mathbb{R}$ and $\mathbb{N}$ to denote the conventional sets of real numbers, and positive integers, respectively. For $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. We use log for the natural logarithm.

Asymptotics

We will often use the following standard asymptotic notations to describe the relative order of magnitude between two sequences x_n and y_n defined on $\mathbb{N}$. For simplicity, we assume that x_n and y_n are nonnegative for all sufficiently large n. We write $x_n = O(y_n)$, or equivalently, $y_n = \Omega(x_n)$ to mean that there is a constant $c > 0$ such that $x_n \leq c y_n$, for all n large enough. If $x_n = O(y_n)$ and $x_n = \Omega(y_n)$, we write $x_n = \Theta(y_n)$. We write $x_n \xrightarrow{n} x$ to mean that x_n converges to x, as n goes to infinity. The statement "$x_n/y_n \xrightarrow{n} 0$" can be rewritten alternatively as $x_n = o(y_n)$,

$y_n = \omega(x_n)$, $x_n \ll y_n$, or $y_n \gg x_n$. We also write $x_n \sim y_n$ to mean that x_n is asymptotic to y_n, that is, $x_n/y_n \xrightarrow{n} 1$.

Probability

We write $\mathbb{P}\{A\}$ for the probability measure of an event A, and $\mathbb{P}\{A|B\}$ for the conditional probability of event A given that event B is true. All the random variables we deal with in this thesis are real-valued measurable functions defined on some probability space. The expected value and the variance of a random variable X are denoted by $\mathbf{E}[X]$ and $\mathbf{Var}[X]$, respectively. The covariance of two random variables X and Y is denoted by $\mathbf{Cov}[X, Y]$. We denote by $\mathbb{I}_{[A]}$ the indicator function of the event A which is 1 if A occurs, and 0 otherwise. For any random variables X and Y, we write $X \stackrel{\mathcal{L}}{=} Y$ to mean that X is distributed as Y, that is, X is equal to Y in law, or $\mathbb{P}\{X \geq t\} = \mathbb{P}\{Y \geq t\}$, for any $t \in \mathbb{R}$.

We say that a sequence of events A_n occurs with high probability (abbreviated w.h.p.), or equivalently, it is true asymptotically almost surely (abbreviated a.a.s.) if and only if $\mathbb{P}\{A_n\} \xrightarrow{n} 1$. Let $X_1, X_2, \ldots$, and X be any random variables. We say that X_n converges in probability to X, as n goes to infinity, if and only if for any constant $\epsilon > 0$, we have

$$\lim_{n\to\infty} \mathbb{P}\{|X_n - X| > \epsilon\} = 0\,.$$

We say that X_n is asymptotic to a_n in probability, where a_n is a real-valued sequence, if X_n/a_n converges to 1 in probability.

0.2 Probabilistic Inequalities

Probabilistic analysis of algorithms is largely about bounding probabilities, especially those of large deviations. Most of the following probabilistic inequalities can be found in any classical probability-oriented textbook, e.g., [7, 97, 137]. See also [100, 116, 123].

Perhaps the simplest probability tail inequality is the one implied by the definition of expectation of any nonnegative random variable X which is

$$\mathbf{E}[X] = \int_0^\infty \mathbb{P}\{X \geq x\}\, dx \geq \mathbb{P}\{X \geq 1\} .$$

This leads to ***Markov's inequality***: for any random variable X, and $t > 0$, we have

$$\mathbb{P}\{|X| \geq t\} \leq \frac{\mathbf{E}[|X|]}{t} .$$

Thus, if f is a nonnegative nondecreasing function defined on $\mathbb{R}$, then for any random variable X, and $t \in \mathbb{R}$,

$$\mathbb{P}\{X \geq t\} = \mathbb{P}\{f(X) \geq f(t)\} \leq \frac{\mathbf{E}[f(X)]}{f(t)} .$$

If we choose $f(x) = x^2$, we obtain ***Chebyshev's inequality***: for any random variable X with bounded mean, and $t > 0$,

$$\mathbb{P}\{|X - \mathbf{E}[X]| \geq t\} \leq \frac{\mathbf{Var}[X]}{t^2} .$$

These bounds are in many cases insufficient. Sharp concentration inequalities can be obtained for random variables that can expressed as functions of independent or almost independent random variables (see below).

Binomial Inequalities

The binomial random variable $\mathrm{Bin}(n, p)$, where $n \in \mathbb{N}$, and $p \in [0, 1]$, has the following distribution:

$$\mathbb{P}\{\mathrm{Bin}(n, p) = k\} = \binom{n}{k} p^k (1-p)^{n-k} , \quad \text{for } k \in \{0, \dots, n\} .$$

Such a random variable can be represented as a sum of n independent binary random variables (or coin flips) where the probability of having 1 is p. The binomial distribution is concentrated around its mean np, i.e., the probability that it deviates from its mean is very small. The following lemma bounds the upper tail probabilities.

Lemma 0.1 (Okamoto [139]). *For $p \in (0,1)$, and $n, t \in \mathbb{N}$, let $\beta := t/n$, and suppose that $n > t > np > 0$. Then*

$$\begin{aligned} \mathbb{P}\left\{\mathrm{Bin}(n,p) \geq t\right\} &\leq \Upsilon(\beta,p)^n \stackrel{\mathrm{def}}{=} \left(\left(\frac{1-p}{1-\beta}\right)^{1-\beta}\left(\frac{p}{\beta}\right)^{\beta}\right)^n & (1) \\ &\leq \left(\frac{epn}{t}\right)^t e^{-pn} & (2) \\ &\leq \left(\frac{epn}{t}\right)^t . & (3) \end{aligned}$$

Observe that if $t = \epsilon np$, for some constant $\epsilon > 1$, then inequality (2) can be written as

$$\mathbb{P}\left\{\mathrm{Bin}(n,p) \geq \epsilon np\right\} \leq \exp\left(-(\epsilon \log \epsilon - \epsilon + 1)np\right) , \tag{4}$$

which is known as ***Angluin-Valiant's inequality*** [11]. All of the above bounds are tight up to a factor of $\Theta(1/\sqrt{n})$, that is, $\mathbb{P}\left\{\mathrm{Bin}(n,p) \geq t\right\} = a_n \Upsilon(\beta,p)^n$, where $a_n = \Theta(1/\sqrt{n})$. Analogous inequalities hold also for the lower binomial tails, but we shall not need them here. The binomial bounds have been implicitly established earlier by Chernoff [31], and were later extended to sums of bounded random variables [16, 11, 88, 93].

The following lemma establishes lower bounds on the upper tail probabilities.

Lemma 0.2. *Let $n, m \in \mathbb{N}$ such that $m/n \xrightarrow{n} \alpha$, for some constant $\alpha \in (0,1)$. Let $p = c/n$, for some constant $c > 0$. Then for $t \in [m-1]$, and n large enough, we have*

$$\mathbb{P}\left\{\mathrm{Bin}(m,p) \geq t\right\} \geq \left(\frac{c\,\alpha}{2t}\right)^t \frac{e^{-c\alpha}}{2} .$$

Proof. Since $(1-1/n)^t \geq 1-t/n$, then for n large enough, we have

$$\binom{n}{t}\frac{1}{n^t} \geq \frac{n}{n-t}\binom{n-1}{t}\frac{(1-1/n)^t}{(n-1)^t} \geq \binom{n-1}{t}\frac{1}{(n-1)^t}\,.$$

By repeating this step, we get $\binom{n}{t}n^{-t} \geq \binom{t}{t}t^{-t} = t^{-t}$. Observe that $(1-c/n)^n \xrightarrow{n} e^{-c}$. Therefore, for n large enough,

$$\begin{aligned} \mathbb{P}\left\{\mathrm{Bin}(n,p) \geq t\right\} &\geq \mathbb{P}\left\{\mathrm{Bin}(n,c/n) \geq t\right\} \\ &\geq \binom{n}{t}\left(\frac{c}{n}\right)^t\left(1-\frac{c}{n}\right)^n \geq \frac{e^{-c}}{2}\left(\frac{c}{t}\right)^t\,. \end{aligned}$$

Since $m/n \xrightarrow{n} \alpha$, then for n large enough, $p \geq c\alpha/(2m)$. Hence, applying the above inequality, we get

$$\mathbb{P}\left\{\mathrm{Bin}(m,p) \geq t\right\} \geq \frac{e^{-c\alpha}}{2}\left(\frac{c\,\alpha}{2t}\right)^t\,.$$

$\square$

Functions with Bounded Differences

The following lemma is useful for bounding complicated random variables that can be written as "nice" functions of independent random variables.

Lemma 0.3 (McDiarmid [122])**.** *Let $X_1,\ldots,X_n$ be independent random variables taking values in a set A, and let f be any real-valued measurable function defined on the set A^n. Suppose that for each $i \in [n]$, there exists $c_i > 0$ such that*

$$\sup_{x_1,\ldots,x_n,\hat{x}_i \in A} |f(x_1,\ldots,x_n) - f(x_1,\ldots,x_{i-1},\hat{x}_i,x_{i+1},\ldots,x_n)| \leq c_i\,,$$

i.e., the function f has bounded differences. Then for any $t \geq 0$, we have

$$\mathbb{P}\left\{f(X_1,\ldots,X_n) - \mathbf{E}\left[\,f(X_1,\ldots,X_n)\,\right] \geq t\right\} \leq \exp\left(\frac{-2t^2}{\sum_{i=1}^n c_i^2}\right)\,,$$

and similarly,

$$\mathbb{P}\left\{f(X_1,\ldots,X_n) - \mathbf{E}\left[\,f(X_1,\ldots,X_n)\,\right] \leq -t\right\} \leq \exp\left(\frac{-2t^2}{\sum_{i=1}^n c_i^2}\right)\,.$$

Notice that the lemma does not require identical distributions for the X_i's. Weaker versions of the lemma for not-totally independent random variables have been also established, see e.g., [122, 123, 116].

Negative Association

The definition of conditional probability says that $\mathbb{P}\{A \cap B\} = \mathbb{P}\{A|B\}\,\mathbb{P}\{B\}$, for any event A and B. Thus, by induction, we see that for any events $A_1, \ldots, A_n$,

$$\mathbb{P}\left\{\bigcap_{i=1}^{n} A_i\right\} = \mathbb{P}\{A_1 \mid A_2, \ldots, A_n\}\,\mathbb{P}\{A_2 \mid A_3, \ldots, A_n\} \cdots \mathbb{P}\{A_{n-1} \mid A_n\}\,\mathbb{P}\{A_n\} .$$

This inequality is useful for studying the maximum value over a set of random values. Plainly, if $X_1, \ldots, X_n$ are random variables, then

$$\begin{aligned}
\mathbb{P}\left\{\max_i X_i \le t\right\} &= \mathbb{P}\{X_1 \le t, \ldots, X_n \le t\} \\
&= \mathbb{P}\{X_1 \le t \mid X_2 \le t, \ldots, X_n \le t\} \cdots \mathbb{P}\{X_n \le t\} .
\end{aligned}$$

Computing the exact probabilities $\mathbb{P}\{X_i \le t \mid X_{i+1} \le t, \ldots, X_n \le t\}$ is usually hard. However, the probabilities can be bounded from above by $\mathbb{P}\{X_i \le t\}$, if the random variables are *negatively associated*, which means that when some of these variables are known to be small, the others are highly unlikely to be small too. The negative association, which is sometimes called negative dependence or correlation, is studied by many researchers, e.g., [56, 57, 63, 68, 98, 111]:

Definition 0.1. Any nonnegative random variables $X_1, \ldots, X_n$ are said to be *negatively associated*, if for every disjoint index subsets $I, J \subseteq [n]$, and for any functions $f : \mathbb{R}^{|I|} \to \mathbb{R}$, and $g : \mathbb{R}^{|J|} \to \mathbb{R}$ that are both non-decreasing or both non-increasing (componentwise), we have $\mathbf{Cov}\left[\, f(X_i, i \in I)\,, g(X_j, j \in J)\,\right] \le 0$, that is,

$$\mathbf{E}\left[\, f(X_i, i \in I)\, g(X_j, j \in J)\,\right] \le \mathbf{E}\left[\, f(X_i, i \in I)\,\right]\, \mathbf{E}\left[\, g(X_j, j \in J)\,\right] .$$

Once we establish that $X_1, \ldots, X_n$ are negatively associated, it follows, by considering inductively the indicator functions, that for any nonnegative numbers $t_1, \ldots, t_n$,

$$\begin{aligned} \mathbb{P}\{X_1 \le t_1, \ldots, X_n \le t_n\} &\le \mathbb{P}\{X_1 \le t_1\}\mathbb{P}\{X_2 \le t_2, \ldots, X_n \le t_n\} \\ &\le \prod_{i=1}^{n} \mathbb{P}\{X_i \le t_i\}, \end{aligned}$$

and similarly,

$$\mathbb{P}\{X_1 \ge t_1, \ldots, X_n \ge t_n\} \le \prod_{i=1}^{n} \mathbb{P}\{X_i \ge t_i\},$$

The next lemmas provide some tools for proving the negative association. For proofs see [63, 98, 57].

Lemma 0.4 (Zero-One Lemma). *Any binary random variables $X_1, \ldots, X_n$ whose sum is one are negatively associated.*

Lemma 0.5. *If $\{X_1, \ldots, X_n\}$ and $\{Y_1, \ldots, Y_m\}$ are independent sets of negatively associated random variables, then the union $\{X_1, \ldots, X_n, Y_1, \ldots, Y_m\}$ is a set of negatively associated random variables.*

Lemma 0.6. *Suppose that $X_1, \ldots, X_n$ are negatively associated. Let $I_1, \ldots, I_k \subseteq [n]$ be disjoint index subsets, for some positive integer k. For $j \in [k]$, let $h_j : \mathbb{R}^{|I_j|} \to \mathbb{R}$ be non-decreasing functions, and define $Z_j = h_j(X_i, i \in I_j)$. Then the random variables $Z_1, \ldots, Z_k$ are negatively associated. In other words, non-decreasing functions of disjoint subsets of negatively associated random variables are also negatively associated. The same holds if h_j are non-increasing functions.*

As an example of negative association we consider the multinomial distribution. Let $X_1, \ldots, X_m$ be independent random numbers chosen from $[n]$ with a common probability distribution, that is, $\mathbb{P}\{X_j = i\} = p_i$, for all $j \in [m]$, and $i \in [n]$, where $p_1 + \cdots + p_n = 1$. For $i \in [n]$, let N_i be the number of times the number i is chosen, i.e., $N_i = \sum_{j=1}^{m} \mathbb{I}_{[X_j=i]}$. The vector $N = (N_1, \ldots, N_n)$ is said to have the multinomial distribution with parameters m and $(p_1, \ldots, p_n)$: for $k_1, \ldots, k_n \in [m]$,

$$\mathbb{P}\{N = (k_1, \ldots, k_n)\} = \frac{m!}{k_1! \cdots k_n!} p_1^{k_1} \cdots p_n^{k_n},$$

if $k_1 + \cdots + k_n = m$, and it is zero, otherwise. The random variables $N_1, \ldots, N_n$ are binomially distributed ($N_i \stackrel{\mathcal{L}}{=} \mathrm{Bin}(m, p_i)$, for all $i \in [n]$), but they are not independent. They are, however, negatively associated. This can be seen by applying Lemma 0.4 to each set of the random variables $\left\{\mathbb{I}_{[X_1=i]}, \ldots, \mathbb{I}_{[X_m=i]}\right\}$, for all $i \in [n]$, and then using Lemmas 0.5 and 0.6. This leads to ***Mallows' inequalities*** [120]:

$$\begin{aligned} \mathbb{P}\{N_1 \le t_1, \ldots, N_n \le t_n\} &\le \prod_{i=1}^{n} \mathbb{P}\{N_i \le t_n\}, \quad \text{and} \\ \mathbb{P}\{N_1 \ge t_1, \ldots, N_n \ge t_n\} &\le \prod_{i=1}^{n} \mathbb{P}\{N_i \ge t_n\}. \end{aligned}$$

Remark 0.1. Sometimes the negative association of random variables can only be proven if a certain event A is true. Generally, we say that the non-negative random variables $X_1, \ldots, X_n$ are negatively associated when conditioned on an event A, if

$$\mathbf{E}\left[f(X_i, i \in I)\, g(X_j, j \in J) \mid A\right] \le \mathbf{E}\left[f(X_i, i \in I) \mid A\right] \mathbf{E}\left[g(X_j, j \in J) \mid A\right].$$

for every disjoint index subsets $I, J \subseteq [n]$, and for any functions $f : \mathbb{R}^{|I|} \to \mathbb{R}$, and $g : \mathbb{R}^{|J|} \to \mathbb{R}$ that are both non-decreasing or both non-increasing (componentwise). One can easily verify that the proofs of Lemmas 0.4, 0.5, and 0.6 are still true when considered with conditioning on some event A. For example, if whenever an event A is true, $X_1, \ldots, X_n$ are binary random variables whose sum is one, then the binary random variables are negatively associated when conditioned on A.

0.3 Allocation Processes

Allocating balls into bins is one of the historical assignment problems [99, 105]. Formally the problem is defined as follows. We are given m balls that have to be placed sequentially into n distinct bins, where each bin can hold an unlimited number of balls. The ***load of a bin*** is defined to be the number of balls it contains. The aim is to design an efficient allocation process that achieves a load distribution on the bins as uniformly as possible. Throughout, we say that an allocation process is ***on-line***, if each ball is assigned upon arrival without knowing anything about the future balls. If the process waits until all the m balls arrive, and considers all the available information about the balls before it places them, then we say that the allocation is ***off-line***.

Remark 0.2. Throughout the thesis, any allocation process is considered to be off-line *if and only if* we mention that *explicitly*, otherwise it is assumed to be on-line.

The balls-and-bins problem is very useful for modelling many applications in computer science such as load balancing, dynamic resource allocation, circuit routing, IP address lookups, and of course hashing. The balls may represent keys, tasks, jobs, users or processes, while the bins may be chains, servers, printers, machines, or processors. For example, in the context of PRAM simulation on distributed memory machines (DMM), we have m processors sharing a memory of PRAM machine that we want to simulate on a DMM machine with n processors and a memory partitioned into n modules, one module per processor. Distributing m balls into n bins means mapping m cells of the shared memory of the PRAM to the n memory modules of the DMM. We shall also explain, in Section 0.4, how the balls and bins can be used to model hashing. For an in-depth view of other applications see [13, 24, 39, 130, 134].

For the remainder of this chapter, we will concentrate on allocation processes that minimize the maximum bin load among all the bins upon their termination.

0.3.1 Classical Allocation Processes

Randomization has been shown to be very effective in minimizing the maximum bin load. For instance, the *classical allocation process* places each ball on-line into a bin chosen independently and uniformly at random, with replacement. Throughout, we shall refer to this process by Classical(n, m), for inserting m balls into n bins.

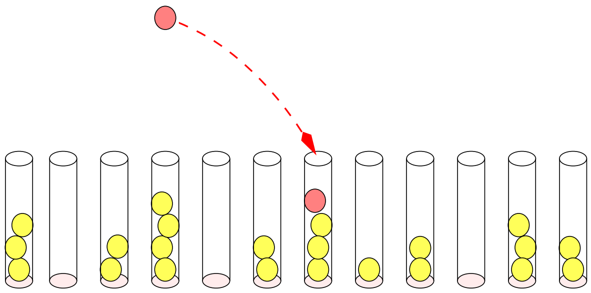

Figure 1: An illustration of Classical(n, m). Each ball is placed in a bin chosen independently and uniformly at random.

The properties of the classical balls-and-bins model (or sometimes called the classical occupancy model) have been extensively analyzed in the probability and statistics literature [99, 105, 100, 107, 20, 75]. Clearly, the load of any bin has the binomial distribution $\mathrm{Bin}(m, 1/n)$, and hence, its expected value is m/n. The bin load vector is multinomial, and by Mallows' inequality, the bin loads are negatively associated. However, it is not difficult to show that each bin load behaves asymptotically as an independent Poisson random variable with parameter m/n.

Theorem 0.1. *Upon termination of* Classical(n, m)*, where* $m = \Theta(n)$*, the maximum bin load among all bins is asymptotic to* $\log n / \log\log n$*, in probability.*

Proofs of the theorem can be found in [99, 105]. In the context of uniform hashing, Gonnet [79] gave another proof based on Poisson approximation which has been simplified by Mitzenmacher [130]. Recently, Raab and Steger [153] presented a new proof using the second moment method, and analyzed the heavily loaded case when $m \gg n$. The process has been also analyzed under assumptions of limited randomness [6, 52, 48, 50, 124], and for nonuniform distributions [41]. Another variation of the classical model has been studied in [55].

0.3.2 Multiple-choice Allocation Processes

The idea of using multiple choices for each ball appears to have been conceived in 1986 in the work of Eager et al. [59]. The power of the idea became more evident in the work of Karp, Luby and Meyer auf der Heide [101]. The authors studied the balls-and-bins problem in the context of PRAM simulation on a distributed memory machine. They allowed each ball to choose two bins independently and uniformly at random, while a simple parallel algorithm decides in which of the two possible bins the ball has to be placed. They proved that if $m = n$, then the allocation process terminates with $O(\log \log n \log^* n)$ maximum bin load, w.h.p.

The Greedy Strategy

In 1994, Azar, Broder, Karlin and Upfal [13] proposed the following novel allocation process. For each ball, we choose $d \geq 2$ bins independently and uniformly at random, with replacement. Then we insert the ball into the least full bin among the d bins, breaking ties randomly. Throughout, we will write Uniform-GreedyMC(n, m, d) to denote this *greedy multiple-choice allocation process* for inserting m balls into n bins. In the case $d = 2$, we may sometimes, for simplicity, omit the third parameter, and just write Uniform-GreedyMC(n, m). The balls are assumed to be inserted on-line and sequentially, unless otherwise explicitly specified.

Notice that the allocation (insertion) time for any ball, (that is, the number of bin accesses) is always d. The maximum bin load of Uniform-GreedyMC(n, m, d),

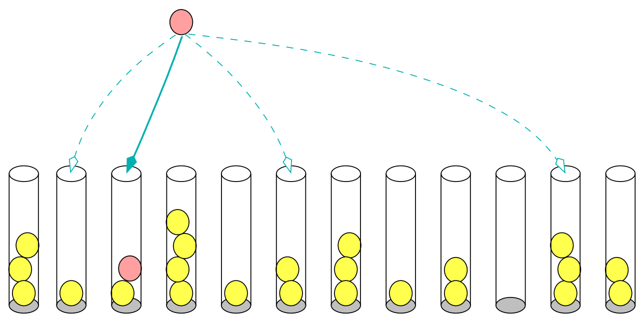

Figure 2: An illustration of UNIFORM-GREEDYMC$(n, m, 4)$. Each ball is inserted into the least loaded bin among 4 bins chosen independently and uniformly at random, with replacement, breaking ties arbitrarily.

surprisingly, decreases significantly , even for $d = 2$. Azar et al. [13] proved that the maximum bin load L_n upon termination of UNIFORM-GREEDYMC(n, n, d) is $\log_d \log n \pm \Theta(1)$, w.h.p., that is, $|L_n - \log_d \log n| \leq c$, w.h.p., for some constant $c > 0$. One can easily generalize these bounds to the case $m = \Theta(n)$. It is also known that the greedy strategy is stochastically optimal in the following sense.

Theorem 0.2 (Azar et al. [13])**.** *Let $n, m, d \in \mathbb{N}$, where $d \geq 2$, and $m = \Theta(n)$. Upon termination of* UNIFORM-GREEDYMC(n, m, d)*, the maximum bin load is* $\log_d \log n \pm \Theta(1)$*, w.h.p. Furthermore, the maximum bin load of any on-line allocation process that inserts m balls sequentially into n bins where each ball is inserted into a bin among d bins chosen independently and uniformly at random, with replacement, is at least* $\log_d \log n - \Theta(1)$*, w.h.p.*

The proof of the above bounds given by Azar et al. [13] uses *the layer induction method*. Mitzenmacher [130, 132] gave another proof based on a system of differential equations called *the fluid limit model*. The upper bound is proved via *the witness tree method* by Vöcking [170]. In Chapter 1, we shall simplify the witness tree argument, and present a new proof for the lower bound. A survey of proof techniques can be found in [134].

The heavily loaded case—when $m = \omega(n)$—of the greedy allocation process has been analyzed by Berenbrink et al. [18]. Using Markov chains, the authors studied the stationary state of the allocation process, and proved the following result.

Theorem 0.3 (Berenbrink et al. [18])**.** *There is a constant $C > 0$ such that for any integers $m \geq n > 0$, and $d \geq 2$, the maximum bin load upon termination of* Uniform-GreedyMC(n, m, d) *is $\log_d \log n + m/n \pm C$, w.h.p.*

Azar et al. [13] also studied an infinite dynamic version of the greedy allocation process Uniform-GreedyMC(n, n, d). Initially, suppose that n balls are inserted by Uniform-GreedyMC, and then at each step a previously inserted ball is selected independently and uniformly at random and removed from the system; and a new ball is inserted into the least bin among d bins chosen independently and uniformly at random, breaking ties randomly. After $\Omega(n^2 \log\log n)$ steps, the maximum bin load still is $\log_d \log n + O(1)$, w.h.p. Vöcking [170] extended the result to any sequence of deletions and insertions that is specified before the algorithm starts. Other infinite dynamic variants of the greedy multiple-choice allocation process are considered in [33, 34].

Theorem 0.4 (Vöcking [170])**.** *Suppose that a possibly infinite sequence of deletions and insertions of balls, that is defined in advance, is performed on-line by algorithm* Uniform-GreedyMC *where each ball is inserted into the least loaded bin among $d \geq 2$ bins chosen independently and uniformly at random from a set of $n \in \mathbb{N}$ bins. Suppose also that at any point of time, there are at most $m = \Omega(n)$ balls in the bins. Then the maximum bin load at any fixed time is $\log_d \log n + O(m/n)$, w.h.p.*

The off-line version of Uniform-GreedyMC$(n, m, 2)$ where the choices available for all balls are known in advance before we insert any ball is studied in [13]. The analysis was further improved by Czumaj and Stemann [39]. A more detailed history of the off-line process is given in Chapter 3.

Asymmetric Variant

Theorem 0.2 asserts that the greedy multiple-choice process is stochastically optimal as long as each ball is inserted on-line into a bin among d bins chosen independently and uniformly at random, with replacement. However, Vöcking [170, 171] demonstrated that it is possible to improve the performance of the greedy process, if nonuniform distributions on the bins and a tie-breaking rule are carefully chosen. He suggested the following variant. First, assume the bins are numbered from 1 to n. Partition the n bins into d groups of almost equal size, that is, each group has size $\Theta(n/d)$. Allow each ball to select upon arrival d bins. All the bins are chosen independently at random where the i-th bin must be chosen uniformly from the i-th group. Each ball is placed on-line, as before, in the least full bin among the d bins. Up to this point, with just these modifications (i.e., the ties are still broken randomly), the maximum bin load, upon termination, is still $\log_d \log n \pm \Theta(1)$, w.h.p.

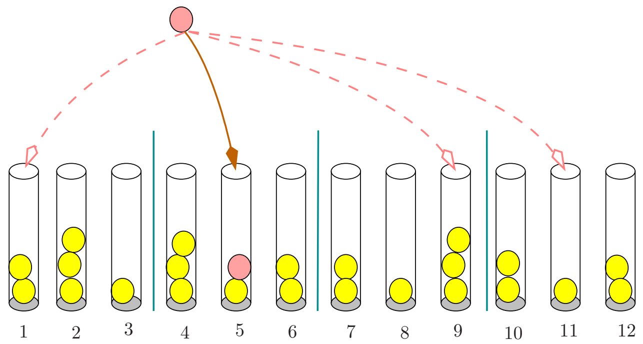

Figure 3: An illustration of LEFTMC$(n, m, 4)$. Each ball is placed in the least full bin among 4 independent bins where the i-th bin is chosen uniformly from the i-th group. Upon a tie, the ball is placed in the leftmost bin.

Vöcking introduced one more crucial change: an asymmetric tie-breaking rule called *Always-Go-Left*. It states that upon a tie, the ball is always placed in the leftmost bin among the d bins. We shall write $\textsc{LeftMC}(n, m, d)$ to refer, throughout, to this variant of the greedy multiple-choice process for inserting m balls into n bins. Vöcking [170] showed that upon termination of $\textsc{LeftMC}(n, n, d)$, the maximum bin load is $\log\log n/(d \log \phi_d) + O(1)$, w.h.p., where ϕ_d is a constant related to a generalized Fibonacci sequence. For example, the constant $\phi_2 = 1.61...$ corresponds to the well-known golden ratio, $\phi_3 = 1.83...$, and $\phi_4 = 1.92....$ In general, $\lim_{d\to\infty} \phi_d = 2$, and $\phi_2 < \phi_3 < \phi_4 < \cdots < 2$. Notice that this is an improvement on the performance of $\textsc{GreedyMc}(n, n, d)$, as $d \log \phi_d > (d - 1) \log 2 > \log d$. For example, when $d = 2$, the maximum bin load of $\textsc{LeftMC}(n, n)$ is $0.72... \times \log_2 \log n + O(1)$, whereas in $\textsc{Uniform-GreedyMC}(n, n)$, it is $\log_2 \log n + O(1)$. The process $\textsc{LeftMC}(n, m, d)$ is also optimal in the following sense.

Theorem 0.5 (Vöcking [170])**.** *Let $n, m, d \in \mathbb{N}$, where $d \geq 2$, and $m = \Theta(n)$. The maximum bin load of* $\textsc{LeftMC}(n, m, d)$ *upon termination is* $\log\log n/(d \log \phi_d) \pm \Theta(1)$*, w.h.p. Moreover, the maximum bin load of any on-line allocation process that inserts m balls sequentially into n bins where each ball is placed into a bin among d bins chosen according to arbitrary, not necessarily independent, probability distributions defined on the bins is at least* $\log\log n/(d \log \phi_d) - \Theta(1)$*, w.h.p.*

Notice that the lower bound on the maximum bin load of $\textsc{LeftMC}(n, m, d)$ holds with *any* probability distributions defined on the bins, and any tie-breaking rule. This is an important result that we shall need in Chapter 2 when we investigate the performance of $\textsc{Uniform-GreedyMC}(n, m)$ with nonuniform distributions. An analogous version of Theorem 0.4 is also proved in [170] confirming that in the dynamic situation, the maximum bin load of $\textsc{LeftMC}(n, m, d)$ does not increase.

The plausible improvement that $\textsc{LeftMC}(n, m, d)$ achieves has been reaffirmed by Mitzenmacher and Vöcking [135, 134] where the process is analyzed in the context of the fluid limit model. Berenbrink et al. [18] studied the heavily loaded case and recorded the following theorem.

Theorem 0.6 (Berenbrink et al. [18]). *There is a constant $C' > 0$ such that for any integers $m \geq n > 0$, and $d \geq 2$, the maximum bin load upon termination of* LeftMC(n, m, d) *is* $\log \log n/(d \log \phi_d) + m/n \pm C'$, *w.h.p.*

Applications and Extensions

A great deal of research has been focused during the last years on analyzing, improving, and generalizing the greedy multiple-choice paradigm. The versatile paradigm has been used to derive efficient algorithms for many applications in computer science. Broder and Mitzenmacher [24] applied the two-way chaining scheme, i.e., algorithm Uniform-GreedyMC(n, m), to improve IP address lookups in internet routers. Byers et al. [26] used the multiple-choice technique to implement distributed hash tables efficiently. The technique has also been utilized in computer graphics [177], routing and interconnection networks [34, 114, 128, 8], queueing systems [130, 131, 135, 172], and shared memory simulations [37, 127].

Many variants and extensions of the greedy multiple-choice process have been introduced and studied in various settings. Mitzenmacher et al. [133] and Shah et al. [159] studied a variant of the greedy multiple-choice process with memory where each time a ball is placed, the least loaded bin of that ball's choices after placement is remembered and used as one of the possible choices for the next ball. The performance of this process is proved to be asymptotically equivalent to LeftMC(n, m, d).

Czumaj and Stemann [39] suggested adaptive multiple-choice allocation processes that achieve optimal trade-offs between the maximum bin load, the maximum allocation time and the average allocation time. For instance, one of the adaptive processes they proposed allows each ball to choose one bin, and inserts the ball into it if its load is at most $\log_d \log n + O(1)$. Otherwise, it chooses $d-1$ more bins and inserts the ball into the least full one. The bins are chosen independently and uniformly at random, with replacement. The authors showed that the maximum bin load upon termination is at most $\log_d \log n + O(1)$, w.h.p., while the maximum allocation time is at most d, and the average allocation time is $1.146194 + o(1)$, w.h.p.

Other studies considered parallel and distributed processes [2, 1, 17, 163], infinite (dynamic) processes with deletion [13, 17, 33, 39, 170], processes that allow re-allocations of the balls [39], and processes with balls of different weights [19].

0.4 Hashing Assumptions

In any hashing scheme (with separate chaining or open addressing) mentioned in the thesis, we insert a set $\mathcal{K}$ of $m \in \mathbb{N}$ distinct input keys, that comes from a finite universe set of keys $\mathcal{U}$, into a table of size $n \in \mathbb{N}$. The keys corresponds to records or data. The hash table is a one-dimensional array with n cells or locations denoted by the set $\mathcal{T} := \{0, \ldots, n-1\}$. In hashing with chaining, each cell in the hash table contains a pointer to a separate chain or linked list. The ***length of a chain*** is defined to the number of keys it contains. The symbol α is reserved, throughout, to denote the ***load factor*** of the hash table m/n. The hashing process uses hash functions that map $\mathcal{U}$ into $\mathcal{T}$. Let $\mathbb{F}(\mathcal{U}, \mathcal{T})$ denote the set of all possible hash functions mapping $\mathcal{U}$ to $\mathcal{T}$. Let $u := |\mathcal{U}|$, and notice that $|\mathbb{F}(\mathcal{U}, \mathcal{T})| = n^u$. We say that a hash function $f : \mathcal{U} \to \mathcal{T}$ is ***truly uniform*** to mean that it is chosen uniformly at random from the set $\mathbb{F}(\mathcal{U}, \mathcal{T})$. Observe that any truly uniform hash function f satisfies the following properties:

1. For any $x \in \mathcal{U}$, the random hashing value $f(x)$ is uniformly distributed over $\mathcal{T}$,

because for any $i \in \mathcal{T}$, we have

$$\mathbb{P}\left\{f(x) = i\right\} = \frac{n^{u-1}}{n^u} = \frac{1}{n}\,.$$

2. The hashing values produced by f are mutually independent (or u-wise independent), because for any distinct $x_1, \ldots, x_k \in \mathcal{U}$, and any $i_1, \ldots, i_k \in \mathcal{T}$, where $k \in [u]$, we have

$$\begin{aligned}\mathbb{P}\left\{f(x_1) = i_1, \ldots, f(x_k) = i_k\right\} &= \frac{n^{u-k}}{n^u} = \frac{1}{n^k} \\ &= \mathbb{P}\left\{f(x_1) = i_1\right\} \cdots \mathbb{P}\left\{f(x_k) = i_k\right\}\,.\end{aligned}$$

One can see now that the classical uniform hashing with chaining where m keys are hashed into n separate chains via only one truly uniform hash function, and which we denote throughout by ClassicChain(n, m), is stochastically equivalent to the classical allocation process Classical(n, m) described above. Similarly, the greedy multiple-choice process Uniform-GreedyMC(n, m) is stochastically equivalent to the uniform two-way chaining scheme, see Chapter 1.

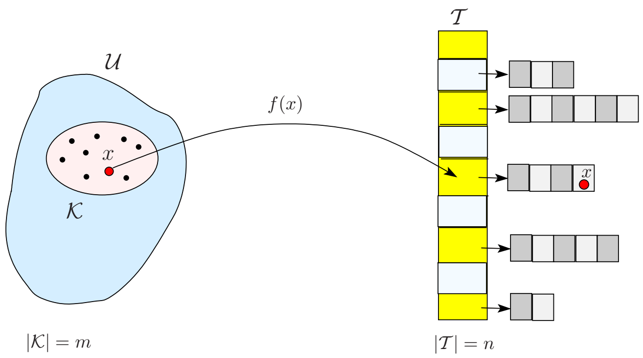

Figure 4: Algorithm ClassicChain(n, m) where keys are hashed by a single truly uniform hash function.

The performance of any hashing algorithm is obviously affected by the complexity of the hash functions it uses. A good hash function is one that can be generated, evaluated, and stored in efficient time and space. The time and space needed to generate a hash function should be at least linear in the hash table size, and the evaluation time should be constant. However, the key probabilistic assumption upon which the mathematical analysis of uniform hashing schemes is built is that the random hash functions used by these schemes are truly uniform. To implement this assumption, we face two practical problems. First, the ability to draw a hash function uniformly at random from the set $\mathbb{F}(\mathcal{U}, \mathcal{T})$ depends heavily on the existence of a pure and true random bit generator, which has not been realized to date. Second, even if we assume the availability of a true source of randomness, the complexity of generating a truly uniform hash function is untractable. Assuming that any key in $\mathcal{U}$ can be represented by one word in $[u]$, i.e., by $\lceil \log_2 u \rceil$ bits, we need $\Theta(u \log n)$ time and space (or number of random bits) to generate and store one truly uniform hash function. That is, the size of the hash function is larger than the size of the table it intends to serve. Notice that the a hash table with $\Theta(n)$ keys consumes only $\Theta(n \log u)$ bits.

Thus, naturally, one wonders if a certain hashing scheme that uses truly uniform hash functions is efficiently realizable and computable in real life in a way that is provable to yield almost the same theoretical performance. The concept of universal hashing, introduced by Carter and Wegman [27, 173], has been proved to be very fruitful in analyzing many hashing schemes under assumptions of limited randomness. The hash functions, there, are drawn uniformly from a smaller family of functions mapping $\mathcal{U}$ to $\mathcal{T}$, instead of the set $\mathbb{F}(\mathcal{U}, \mathcal{T})$. Although, the hashing values of such functions are almost uniform and almost k-wise independent, for some $k \ll n$, they are sufficient enough to give almost the same performance of truly uniform hash functions [52, 48, 50, 161].

In this thesis, however, and for simplicity, all the hashing schemes are studied with truly uniform hash functions, except in Chapter 2 where we analyzed two-way chaining with nonuniform hash functions that satisfy only Property 2. We also assume that the hashing schemes are implemented on a RAM model of computation where memory access and the standard arithmetic and logic operations can be executed in one unit time; in particular, probing or accessing a hash table cell can be done in one unit time. Furthermore, we assume that a pure random source is available, that is, it is feasible to choose objects uniformly at random. This assumption, in particular, is reasonable, as high quality pseudo-random bit generators are readily available.

We define the ***search time*** of any hashing algorithm as the number of probes or table accesses the algorithm performs to find a key. Observe that we *ignore* the time required to evaluate the hash functions. In particular, we define the search time in algorithm ClassicChain(n, m) to be one (for accessing the pointer to the chain) plus the number of keys the algorithm examines. For example, in Figure 4, the time needed to search for the key x according to our definition is 5. Similarly, the ***insertion*** or ***deletion time*** is defined to be the number of probes the algorithm performs to insert or delete a key, respectively.

Finally, notice that any hashing scheme can be classified as on-line or off-line just as we classify any allocation process. That is, a hashing scheme is said to be on-line, if each key is hashed upon arrival without knowing any information about the future keys. It is said to be off-line if the hashing values available for all keys are known in advance before any insertion. Throughout the thesis, all hashing schemes are assumed to be on-line, unless we explicitly mention otherwise. Moreover, hashing schemes can be also described as ***static*** when the hashing data are not allowed to be updated or deleted. Otherwise, the hashing scheme is said to be ***dynamic***.

Part I

Hashing with Separate Chaining

Hashing emerges as a heuristic technique for supporting dictionary operations to store and retrieve information in constant expected time. In hashing with separate chaining, keys that collide in the same cell are inserted into a separate chain (or linked list) pointed to by the cell. According to Knuth [103], hashing with chaining seems to have been originated by H. P. Luhn in an internal IBM memorandum in 1953. However, Dumey [58], in 1956, was the first to describe the technique in the open literature. Since then, various hashing schemes with different collision resolutions have been invented and analyzed. Most notably are open addressing schemes [147], coalesced hashing [167, 168], extendible hashing [65], linear and dynamic hashing [108, 113, 110], perfect hashing [74, 50, 144], universal hashing [27, 173], cuckoo hashing [146, 69], and, of course, two-way chaining [13, 170]. Comparison-based or tree-oriented data structures are also suggested for implementing dictionaries. However, their expected performance is slow, especially, when the data structure is updated. A wealth of information about these methods and others can be found in [103, 121, 80, 169, 145].

This part of the thesis is devoted only to the two-way chaining paradigm. We study the on-line uniform version in Chapter 1, and the on-line nonuniform case in Chapter 2. We analyze the off-line uniform version in Chapter 3. Chapter 4 contains a discussion on some trade-offs and speedups of this hashing paradigm.

Chapter 1

Uniform Two-way Chaining

We begin our study by presenting another proof of the first part of Theorem 0.2 concerning the worst-case performance of the greedy multiple-choice allocation process Uniform-GreedyMC(n, m, d). We only consider the case $d = 2$. We shall see in Chapter 4 that the best worst-case performance, however, is achieved when $d = 3$. We choose the case $d = 2$ for the sake of simplicity, and because its average-case performance is better than the case $d = 3$.

1.1 History and Motivation

Dictionaries are fundamental data structures designed specially for storing data and supporting basic operations like insert, delete and search. Dictionaries can be static or dynamic. In static dictionaries, the data structures are not allowed to be updated. Hashing emerges as a very efficient technique for implementing dictionaries. For example, algorithm ClassicChain(n, m), the classical uniform hashing with separate chaining, is widely known for its simplicity and its plausible average performance. Indeed, the expected average successful search time is $1 + \alpha/2$, and the expected unsuccessful search time is $1 + \alpha$, where $\alpha := m/n$, see [80, 35, 169]. Unfortunately, the worst-case unsuccessful search time which is proportional to the length of the longest chain is proved in [79] to be asymptotic to $\log n/\log\log n$, in probability, when $m = \Theta(n)$. This is also a direct application of Theorem 0.1, as the classical allocation process $\textsc{Classical}(n, m)$ is stochastically equivalent to this hashing scheme.

Carter and Wegman [27, 173] suggested the concept of universal hashing as a theoretical framework for analyzing classical chaining with more practical hash functions. They showed that the asymptotic average performance of $\textsc{ClassicChain}(n, m)$ can be almost preserved (up to a constant factor), if we choose the hash function uniformly at random from a smaller class of functions mapping the universe set of keys $\mathcal{U}$ to the hash table $\mathcal{T}$. The class of functions can be designed to be small and contains only efficient hash functions that can be generated in linear time and space, and evaluated in constant time. Many such classes have been designed, see e.g., [6, 48, 52, 53, 141, 161]. Nonetheless, none of these classes lead to a better worst-case performance (when used in classical hashing with separate chaining) than the one achieved by a truly uniform hush function.

During the last two decades, an intensive research has been concentrated on improving the worst-case search time, and consequently, many randomized hashing schemes (with or without chaining) have been introduced. We survey the most prominent ones.

Randomized Perfect Hashing

A perfect hash function on a subset of keys $\mathcal{K} \subseteq \mathcal{U}$ is a 1-1 function that maps $\mathcal{K}$ to the hash table $\mathcal{T}$. A perfect hashing algorithm is an algorithm that inserts the keys without any collisions. Thus, the worst-case search time can be dramatically decreased, if the perfect hash functions used by the algorithm are constructed in an efficient way. Notice that by the birthday paradox, any randomly chosen hash function is perfect only on some subsets of keys, but not on every subset, unless the size of $\mathcal{T}$ is very large. Thus, the challenge is to design efficient perfect hashing schemes for hash tables of linear size, i.e., $|\mathcal{T}| = O(|\mathcal{K}|)$. Many such schemes have been introduced [74, 36, 50, 142, 144] with efficient perfect hash functions that can be evaluated in constant time and constructed in expected linear time and space. For example, Fredman et al. [74] presented a randomized perfect hashing algorithm that inserts n keys into a hash table of size $n + o(n)$, and achieves constant maximum search time, and constant expected amortized insertion time. The hashing algorithm, however, is off-line and static. The algorithm first uses a hash function chosen randomly from a small class of functions to partition the set of input keys into $\Theta(n)$ disjoint groups. Each group, then, is hashed to a separate sub-table by a perfect hash function chosen randomly from a set of functions designed specifically for that group.

The static hashing algorithm has been generalized by Dietzfelbinger et al. [50] to the dynamic situation where updates and deletions are allowed, while preserving almost the same performance. Similarly, they used a random hash function to partition the keys into $\Theta(n)$ disjoint groups, and a different perfect hash function to hash each group. However, to cope with the dynamic data, they used a rehashing technique where the whole hash table is reconstructed from the beginning by using new random hash functions whenever the number of updates exceeds certain limit. The worst-case search time, and the expected amortized insertion and deletion times are still constants, but the storage space consumed by the hash table is $35(1+c)n$, where $c > 0$ is a constant. The update performance of this scheme was further improved in Dietzfelbinger and Meyer auf der Heide [51, 52], where a new efficient universal class is used to achieve constant worst-case time for any dictionary operation, with high probability. Notice that all of these schemes employ $\Theta(n)$ random hash functions that require a large number of random bits. Dietzfelbinger et al. [48] described how one can reduce the number of random bits consumed by these schemes by using polynomial hash functions. For a more detailed study of perfect hashing, see [36, 145].

Open Addressing Schemes

Many open addressing schemes with improved worst-case performance are based, more or less, on multilevel hashing where the hash table is partitioned into multiple sub-tables, and different hash functions are used for each sub-table. For instance, Broder and Karlin [23] divide the hash table into $\Theta(\log\log n)$ blocks, and with each block, they use a different hash function chosen randomly form a universal class of functions. The first hash function is used to insert each key into the first block. If a collision happens, then the key is hashed by the second hash function into the second block. If a collision occurs again, then the third block is checked by the third hash function, and so on. If a collision occurs in the last block, then a rehashing technique is used. The expected amortized time for insertion, deletion or search is constant, but the worst-case search time is $\Theta(\log\log n)$, deterministically. Of course, if parallel computations including memory accesses and hash function evaluations are allowed, then any instruction can be executed in constant time.

Most recently, Pagh and Rodler [146] introduced *cuckoo hashing*. They insert n keys into a hash table that is partitioned into two parts, each of size $\lceil (1+\epsilon)n \rceil$, for some constant $\epsilon > 0$. It uses two independent hash functions chosen from an $O(\log n)$-universal class—one function only for each sub-table. Each key is hashed initially by the first function to a cell in the first sub-table. If the cell is full, then the new key is inserted there anyway, and the old key is kicked out to the second sub-table to be hashed by the second function. The same rule is applied in the second sub-table. Keys are moved back and forth until a key moves to an empty location or a limit of $O(\log n)$ moves is reached. When the limit is reached, new independent hash functions are chosen, and the whole table is rehashed. The worst-case search time is at most two, and the amortized expected insertion time, nonetheless, is constant. An off-line and static version of this algorithm had previously appeared in [144]. Other analyses and extensions of this scheme can be found in [44, 69, 53, 141].

Deterministic Dictionaries

Deterministic methods for implementing dictionaries that do not use random bits include perfect hashing, and comparison-based or tree-oriented data structures (see e.g., [10, 129, 86, 143, 145] and the references cited there). All of these techniques, however, require $\omega(\log \log n)$ time either for searching, or for updating and maintaining the data structure. For example, Andersson [10] designed a deterministic dictionary that can be constructed in linear time and space, but the worst-case search time is $\Omega(\log n)$. On the other hand, Hagerup et al. [86] presented a deterministic dictionary that has constant worst-case search time, but the insertion time is $\Omega(\log n)$. Pagh [143] considered a compromised deterministic dictionary where the search time is $(\log \log n)^{O(1)}$, and the update time is $(\log n)^{O(1)}$. Needless to say that in some of the comparison-based data structures such as balanced trees, the worst-case cost for any operation is $\Omega(\log n)$.

1.2 Two-way Chaining

The two-way chaining paradigm suggested by Azar et al. [13] is a simple approach for dramatically improving the worst-case search time of hashing with chaining. To avoid any ambiguity, we define it formally as follows. Recall that we denote the universe set of keys by $\mathcal{U}$, and the hash table by $\mathcal{T}$. The cells of the hash table are numbered, and each cell points to a separate chain or linked list. For simplicity, we will say "the chain i" to mean the chain that is pointed to by the cell i. The length of a chain is defined to be the number of keys it contains. We assume, throughout, that we save with each cell in the hash table the length of the chain that the cell points to, and we keep it updated.

Definition 1.1. An on-line two-way chaining algorithm is an algorithm that satisfies the following:

1. It inserts a set of keys $\mathcal{K} \subseteq \mathcal{U}$ sequentially (one after another) into a hash table $\mathcal{T}$ where collisions are resolved by separate chaining.

2. It uses two hash functions $f, g : \mathcal{U} \to \mathcal{T}$.

3. Each key $x \in \mathcal{K}$ is inserted into the shortest chain (i.e., with the least number of keys) among the two chains $f(x)$ and $g(x)$, where ties are broken according to some fixed strategy.

Clearly, the insertion time of any two-way chaining algorithm is constant. To search for any key x, we examine the two chains $f(x)$ and $g(x)$, sequentially and alternatingly. Thus, the maximum unsuccessful search time is proportional to twice the length of the longest chain. Trivially, the performance of any two-way chaining algorithm depends on the type of the hash functions and the tie-breaking rule it uses.

Throughout the thesis, we write Nonuniform-ShortChain to denote the on-line two-way chaining algorithm that satisfies the following:

A. It uses two independent random hash functions f and g, i.e., $f(x)$ and $g(x)$ are random variables with two independent probability distributions defined on $\mathcal{T}$.

B. If for some key x, both chains $f(x)$ and $g(x)$ have the same length, then the algorithm breaks the tie randomly.

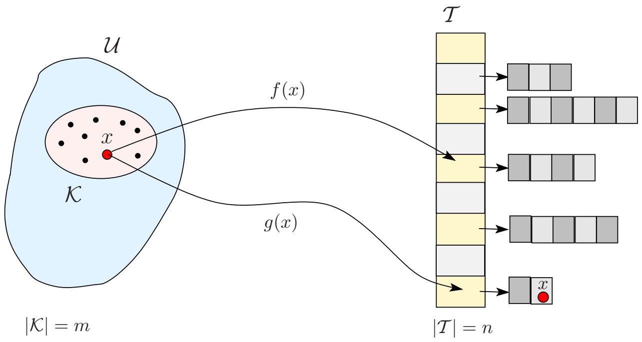

Figure 1.1: Algorithm UNIFORM-SHORTCHAIN(n, m) where f and g are independent and truly uniform hash functions.

If, moreover, the hash functions f and g are independent and truly uniform, then we write UNIFORM-SHORTCHAIN. This means that f and g are chosen independently and uniformly at random from the set of all possible hash functions $\mathbb{F}(\mathcal{U}, \mathcal{T})$. We often write NONUNIFORM-SHORTCHAIN(n, m) or UNIFORM-SHORTCHAIN(n, m), for $n, m \in \mathbb{N}$, to mean that the algorithms insert a set of keys $\mathcal{K}$ into the hash table $\mathcal{T}$, where $|\mathcal{K}| = m$, and $|\mathcal{T}| = n$.

Observe that algorithm UNIFORM-SHORTCHAIN(n, m) is stochastically equivalent to the greedy multiple choice allocation process UNIFORM-GREEDYMC(n, m). Thus, by Theorems 0.2 and 0.3, the maximum chain length is $\log_2 \log n + m/n \pm \Theta(1)$, w.h.p., for $m = \Omega(n)$. On the other hand, the following theorem states that the average search time of algorithm UNIFORM-SHORTCHAIN(n, m) is not more than twice the average search time of the classical uniform chaining algorithm CLASSICCHAIN(n, m).

Theorem 1.1 (Azar et al. [13]). *Let $n, m \in \mathbb{N}$. The average expected successful search time of algorithm* UNIFORM-SHORTCHAIN(n, m) *is at most $2 + m/n$, and the average unsuccessful search time is at most $2 + 2m/n$.*

One can also mimic the multiple choice allocation process LEFTMC(n, m) designed by Vöcking [170, 171] to derive a two-way chaining algorithm. Indeed, we shall write LEFT-SHORTCHAIN to denote the on-line two-way chaining algorithm that satisfies the following:

A. The hash functions f and g are chosen independently and uniformly at random from the sets $\mathbb{F}(\mathcal{U}, \mathcal{T}_1)$, and $\mathbb{F}(\mathcal{U}, \mathcal{T}_2)$, respectively, where $\mathcal{T}_1 := \{0, \ldots, \lfloor n/2 \rfloor\}$, and $\mathcal{T}_2 := \{\lceil n/2 \rceil, \ldots, n-1\}$ constitute a partition of the hash table.

B. If for some key x, the chains $f(x)$ and $g(x)$ have the same length, the key is inserted into the chain $f(x)$.

The notation LEFT-SHORTCHAIN(n, m) has the same meaning as we explain above for other algorithms. Similarly, algorithm LEFT-SHORTCHAIN(n, m) is stochastically equivalent to LEFTMC(n, m), and by Theorems 0.5 and 0.6, the maximum chain length is $0.72... \times \log_2 \log n + m/n \pm \Theta(1)$, w.h.p., when $m = \Omega(n)$. The average search time is at worst twice the average search time of CLASSICCHAIN(n, m).

It is evident that these two-way chaining algorithms reduce, stochastically and asymptotically, the worst-case performance exponentially—comparing to classical chaining method—at the expense of doubling the average case performance. Two-way chaining also has several advantages over the other hashing methods we mentioned above for improving the worst-case behavior of hashing. Clearly, it is on-line and dynamic, it employs only two hash functions, it is easy to parallelize, and it does not use any rehashing technique. Unlike most of the above schemes, its worst-case insertion time is still constant. It consumes almost the same storage space as classical chaining. Note that the additional memory space is needed only to store at worst n integers which corresponds to the chain lengths where each one consumes at most $O(\log \log \log n)$ bits, w.h.p. Furthermore, the same hashing performance is provably achievable even if the hash functions are chosen from a smaller class of hash functions, e.g., an $O(\log n)$-universal class, like the ones in [48, 101].

The two-way chaining paradigm has been effectively used to derive many efficient algorithms for various applications [24, 26, 114, 37, 177]. As we mentioned earlier, the use of two hash functions appeared previously in [59, 101], and later in [146]. For more related history and other applications see Section 0.3.2.

In Chapter 2, we present a stochastic analysis of the worst-case performance of algorithm NONUNIFORM-SHORTCHAIN(n, m), where $n, m \in \mathbb{N}$ and the two used hash functions are possibly nonuniform. Observe that the hash functions used in LEFT-SHORTCHAIN are also nonuniform. For the remainder of this chapter, however, we will concentrate on the worst-case performance of the uniform two-way chaining when the load factor is 1, that is, algorithm UNIFORM-SHORTCHAIN(n, n).

Theorem 1.2. *Upon termination of algorithm* UNIFORM-SHORTCHAIN(n, n)*, where* $n \in \mathbb{N}$*, the maximum (unsuccessful or successful) search time is* $2 \log_2 \log n \pm \Theta(1)$*, w.h.p.*

It is worth noting that many techniques are used to analyze the worst-case performance of two-way chaining algorithms. Azar et al. [13] used the *layer induction method* to bound the maximum chain length of UNIFORM-SHORTCHAIN(n, m). Mitzenmacher [130, 132] used a system of differential equations in his *fluid limit model*. Using a method called *witness trees*, Vöcking [170, 171] studied the worst-case performance of algorithms UNIFORM-SHORTCHAIN(n, m) and LEFT-SHORTCHAIN(n, m). Berenbrink et al. [18] utilized *coupling methods of Markov chains* to investigate the heavily loaded case (where $m \gg n$) of both of these algorithms. See also [134] for a brief explanation of these techniques.

We prove, in the next section, the lower bound stated in Theorem 1.2 by using a *waiting time argument*. In Section 1.4, we use a simpler version of the witness tree method to prove the matching upper bound.

1.3 The Lower Bound

Recall that the time needed to search for any key x by the hashing algorithm Uniform-ShortChain is defined to be the number of keys visited during the search operation plus two for reading the two head-pointers to the two chains $f(x)$ and $g(x)$, where f and g are the hash functions used by the algorithm. Notice that if y is the last key inserted into a chain of maximum length, then the difference between the lengths of the chains $f(y)$ and $g(y)$ is at most one. Thus, the worst-case (unsuccessful or successful) search time is equal to twice the maximum chain length plus constant. Since the maximum chain length in algorithm Uniform-ShortChain(n, n) is distributed as the maximum bin load in the greedy multiple-choice allocation process Uniform-GreedyMC(n, n), it suffices to prove the following theorem.

Theorem 1.3. *For $n \in \mathbb{N}$, let L_n be the maximum bin load upon termination of algorithm* Uniform-GreedyMC(n, n)*. Then $L_n \geq \log_2 \log_2 n - \Theta(1)$, w.h.p.*

Proof. Recall that the allocation process Uniform-GreedyMC(n, n) inserts n balls sequentially into n bins, where each ball is inserted into the least full bin among two bins chosen independently and uniformly at random, breaking ties randomly. The following proof uses a waiting time argument that divides the allocation process into multiple stages. Initially, suppose we have a set of $n_0 > 0$ bins that we call *the set of the initial survival bins.* At each stage we refine these survival bins by selecting some of them, until we reach the stage where we have only one survival bin at which we stop. An initial survival bin *survives* the first stage if and only if a ball is inserted into it during the first stage; and for all $k \geq 2$, a survival bin of the $(k-1)$-th stage *survives* the k-th stage if it satisfies one of the following conditions:

1. The bin contains at least k balls before it is chosen by a ball (as one of its two choices) during the k-th stage.

2. The bin contains $k-1$ balls before it is chosen by a ball which is inserted into it during the k-th stage.

Observe that for all $k \geq 1$, any bin that survives the k-th stage has at least k balls in it. In the two conditions above, we say that the ball *helps* the bin to survive. We say that the *survival time* of a survival bin of the k-th stage is t, if the ball that helps it to survive the k-th stage is the t-th ball inserted during the k-th stage. The first stage starts with the first ball we insert. Then we keep inserting balls sequentially, and wait until $n_1 < n_0/2$ bins from the initial survival bins survive the first stage, at which it finishes. Then the second stage starts by inserting more balls sequentially, and it finishes once $n_2 < n_1/2$ bins survive, and so on. In the k-th stage, we wait until $n_k < n_{k-1}/2$ bins survive. The sequence n_k will be picked later on. Let T_k be the number of balls inserted during the k-th stage. That is, T_k is the survival time of the last survival bin of the k-th stage. Now if r is our lower bound and assuming $n_r \geq 1$, we only need to show that $\sum_{k=1}^{r} T_k \leq n$, w.h.p. In other words, the number of balls that we should insert to reach the r-th level is, asymptotically almost surely, not more than n.

Consider only the k-th stage. Let A_t be the event that the t-th ball inserted during the k-th stage helps a bin to survive. Let $\mathcal{H}_t$ be the history up to time t. Recall that the load of any survival bin of the $(k-1)$-th stage is at least $k-1$. So, if the t-th ball chooses two survival bins of the $(k-1)$-th stage that have not survived the k-th stage yet (and there are at least $n_{k-1} - n_k$ such bins), then the t-th ball helps at least one bin to survive the k-th stage. Since the bins are chosen independently and uniformly at random, and $n_k < n_{k-1}/2$, then we have

$$\mathbb{P}\{A_t \mid \mathcal{H}_{t-1}\} \geq \left(\frac{n_{k-1} - n_k}{n}\right)^2 > \left(\frac{n_{k-1}}{2n}\right)^2 \stackrel{\text{def}}{=} \rho_k \,.$$

Let S_j be the survival time of the j-th survival bin of the k-th stage. Clearly, $S_1 > t$ if and only if the first t balls inserted during the k-th stage did not help any bin to survive. By using conditional probabilities, we see that

$$\begin{aligned}\mathbb{P}\{S_1 > t\} &= \mathbb{P}\{A_1^c\}\,\mathbb{P}\{A_2^c \mid A_1^c\}\cdots\mathbb{P}\left\{A_t^c \mid \cap_{j=1}^{t-1} A_j^c\right\} \\ &< (1-\rho_k)^t \le e^{-t\rho_k} = \mathbb{P}\{E/\rho_k > t\}\ ,\end{aligned}$$

where E is an exponential random variable with density $h(x) = e^{-x}$ on $[0,\infty)$. Note that $\mathbf{E}[\,E\,] = \mathbf{Var}[\,E\,] = 1$, see e.g., [83]. Since the above chain of inequalities is valid for all t, we say that $S_1 \prec E/\rho_k$: S_1 is *stochastically smaller* than E/ρ_k. Switching to exponential random variables helps us to bound the variable T_k. Recall that T_k is the survival time of the last survival bin of the k-th stage, i.e., $T_k = S_{n_k}$. However, $S_{n_k} = S_1 + (S_2 - S_1) + \cdots + (S_{n_k} - S_{n_k-1})$, and each difference $S_j - S_{j-1}$ is stochastically smaller than E/ρ_k. Thus, if $G_{n_k} := \sum_{j=1}^{n_k} E_j$, where $E_1, \ldots, E_{n_k}$ are independent exponential random variables, then $T_k = S_{n_k} \prec G_{n_k}/\rho_k$. Observe that $\mathbf{E}[\,G_{n_k}/\rho_k\,] = n_k/\rho_k$, and $\mathbf{Var}[\,G_{n_k}/\rho_k\,] = n_k/\rho_k^2$, because the E_j are independent. Therefore, we have the following probabilistic duality

$$\mathbb{P}\{L_n < r\} \le \mathbb{P}\left\{\sum_{k=1}^{r} T_k > n\right\} \le \mathbb{P}\left\{\sum_{k=1}^{r} \frac{G_{n_k}}{\rho_k} > n\right\}\ .$$

For simplicity, let $Z_r = \sum\limits_{k=1}^{r} G_{n_k}/\rho_k$, and notice that

$$\mathbf{E}[\,Z_r\,] = \sum_{k=1}^{r} \frac{n_k}{\rho_k} = \sum_{k=1}^{r} \frac{4 n_k n^2}{n_{k-1}^2}\ ,$$

and

$$\mathbf{Var}[\,Z_r\,] = \sum_{k=1}^{r} \frac{n_k}{\rho_k^2} \le \frac{1}{\rho_r} \sum_{k=1}^{r} \frac{n_k}{\rho_k} = \frac{4n^2}{n_{r-1}^2}\mathbf{E}[\,Z_r\,]\ .$$

Now for n large enough, we define $n_0 := n$, and for $k \ge 1$,

$$n_k := \left\lfloor n 2^{k+\kappa-1} \Big/ 2^{2^{k+\kappa}} \right\rfloor\ ,$$

where $\kappa > 1$ is an integer to be chosen later. There are two reasons for using κ: one of them is to make sure that $n_k < n_{k-1}/2$, for all $k \ge 1$, and hence κ must be at least

2. The other reason will be clarified soon. Notice that for all $k > 0$, we have

$$n_k \leq \frac{n 2^{k+\kappa-1}}{2^{2^{k+\kappa}}}, \quad \text{and} \quad n_{k-1}^2 \geq \frac{n^2 2^{2(k+\kappa)}}{2^6 \cdot 2^{2^{k+\kappa}}},$$

Also, if we put $r := \lceil \log_2 \log_2 n - \kappa - 1 \rceil$, we see that $n_r \geq 1$, for n large enough, and $n_{r-1}^2 \geq 2^{-8}\, n (\log_2 n)^2$. Thence,

$$\mathbf{E}\left[\, Z_r \,\right] = 4 \sum_{k=1}^{r} \frac{n_k n^2}{n_{k-1}^2} \leq 2^7 \sum_{k=1}^{r} \frac{n}{2^{k+\kappa}} \leq \frac{2^7 n}{2^\kappa} = \frac{n}{2},$$

which is true if we set $\kappa := 8$. Hence, we get $\mathbf{Var}\left[\, Z_r \,\right] \leq 2^9 n^2 (\log_2 n)^{-2} = o(n^2)$. Using Chebyshev's inequality, we conclude that

$$\mathbb{P}\left\{L_n < r\right\} \leq \mathbb{P}\left\{Z_r > n\right\} \leq \mathbb{P}\left\{Z_r - \mathbf{E}\left[\, Z_r \,\right] > n/2\right\} \leq 4\, \mathbf{Var}\left[\, Z_r \,\right] / n^2 = o(1)\,.$$

□

Remark 1.1. In dynamic hashing, keys (or balls) are allowed to be deleted or updated. Clearly, the above proof is not valid if we consider any arbitrary sequence of insertions and deletions of balls $\omega_1, \omega_2, \omega_3, \ldots$, where ω_t is the t-th request to be performed by algorithm UNIFORM-GREEDYMC which inserts each ball into the least loaded bin among two bins chosen independently and uniformly at random from a set of $n \in \mathbb{N}$ bins. This is mainly because the number of balls at any level may decrease or increase with time, and the insertion of the future coming balls, obviously, depends on the distribution of the balls that still reside in the bins. However, assuming that the sequence of requests is specified in advance, that is, independently of the decisions made by the algorithm, one can show that the maximum bin load at a fixed time t is still at least $\log_2 \log n - \Theta(1)$, w.h.p., provided that there exist at least $\Omega(n)$ balls in the bins at that time. For example, suppose that $\omega_1, \ldots, \omega_n$ are requests for insertion of balls, and ω_t, for $t > n$ is an insertion or a deletion request such that at any time t there are at least $\Omega(n)$ balls that reside in the bins. Then considering only the balls that still exist in the bins, one can see that each one of these balls has been inserted on-line (without using any information about the balls that are inserted after it) into a bin among two bins chosen independently and uniformly at random. Therefore, by the second part of Theorem 0.2, the maximum bin load is at least $\log_2 \log n - \Theta(1)$, w.h.p.

1.4 The Upper Bound

Theorem 1.4. *For $n \in \mathbb{N}$, let L_n be the maximum bin load upon termination of algorithm* Uniform-GreedyMC(n, n)*. Then $L_n \leq \log_2 \log_2 n + \Theta(1)$, w.h.p.*

We prove this theorem by using the witness tree method which has appeared in many studies, see e.g., [33, 34, 127, 134, 157, 170]. The proof we provide here is similar to the one used in [157, 170], but it is simpler, shorter, and clearer. We show that if there exists a bin with at least h balls, then there is a witness tree of height h that describes the history of that bin, and the probability that such tree occurs tends to zero, as n goes to infinity, when h is sufficiently large. The formal definition of a witness tree is given below. Recall that the balls are inserted sequentially into the bins where each ball is placed into the least loaded bin among two bins chosen independently and uniformly at random, breaking ties randomly. Throughout, we assume the balls are numbered $1, \ldots, n$ according to their insertion time. For each $t \in [n]$, we write X_t and Y_t to denote the first and the second choices of bins available for ball t, i.e., the t-th inserted ball. We shall first define the history tree of a ball which could be full or truncated. A witness tree is a special truncated history tree.

The History Tree

We define for each ball t a *full history tree* T_t, which is a deterministic colored binary tree that is labelled by ball numbers except possibly the leaves. Each ball is identified with the bin that contains it. So the full history tree T_t, indeed, describes the history of the bin that contains the t-th ball up to its insertion time. It is a binary tree that represents all the pairs of bins available for all other balls upon which the final position of the ball t relies. Formally, we define it as follows. The root of T_t is labelled t, and is colored white. The root has two children, a left child corresponding to the bin X_t, and a right child corresponding to the bin Y_t. The left child is labelled and colored according to the following rules:

(a) If the bin X_t contains some balls at the time of insertion of ball t, and the last ball inserted in that bin, say τ, has not been encountered thus far in the Breadth-First-Search (BFS) order of the binary tree T_t, then the node is labelled τ and colored white.

(b) As in case (a), except that τ has already been encountered in the BFS order. We distinguish such nodes by coloring them black, but they get the same label τ.

(c) If the bin X_t is empty at the time of insertion of ball t, then it is a "dead end" node without any label and it is colored gray.

Similarly, the right child of t is labelled and colored by following the same rules but with the bin Y_t. We continue processing nodes in BFS fashion. A black or gray node in the tree is a leaf and is not processed any further. A white node with label τ is processed in the same way we processed the ball t, but with its two bins X_τ and Y_τ. We continue recursively constructing the tree until all the leaves are black or gray. See Figure 1.2 for an example of a full history tree.

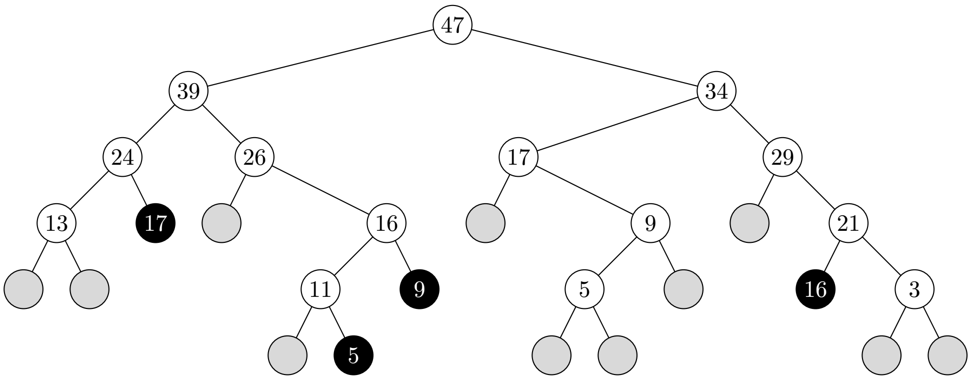

Figure 1.2: The full history tree of ball 47. White nodes represent type (a) nodes. Black nodes are type (b) nodes—they refer to balls already encountered in BFS order. Gray nodes are type (c) nodes—they occur when a ball selects an empty bin.

Note that every internal (white) node of the full history tree has two children. Furthermore, there is at least one gray leaf. Also, since the insertion process is sequential, node values (ball numbers) along any path down from the root must be decreasing (so the binary tree has the heap property), because any non-gray child of any node represents the last ball inserted in the bin containing it at the insertion time of the parent. We will not use the heap property however.

It is clear that the full history tree permits one to deduce the load of the bin that contains the root ball at the time of its insertion: it is the length of the shortest path from the root to any gray node, where the length of a path is defined to be the number of edges in it. Thus, if the bin's load is more than h, then all gray nodes must be at distance more than h from the root, that is, all the first $h+1$ levels do not contain any gray node. This leads to the notion of a ***truncated history tree*** of height h, that is, with $h+1$ levels of nodes. The top part of the full history tree T_t that includes all nodes at the first $h+1$ levels is saved, and the remainder is truncated, see Figure 1.3.

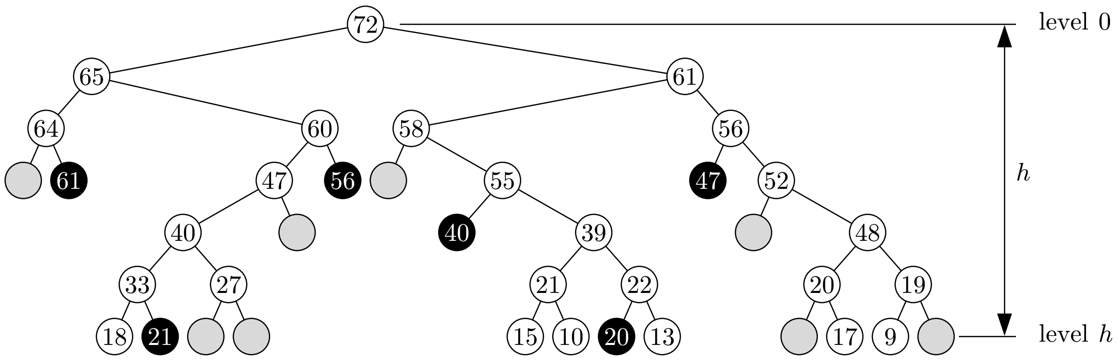

Figure 1.3: A truncated history tree of height h for ball 72.

We are in particular interested in truncated history trees of height h that do not contain any gray nodes. Thus, by the property mentioned above, the length of the shortest path from the root to any gray node in the full history tree (and as noted above, there is at least one such node) would have to be at least $h+1$. Therefore, the load of the bin harboring the root's ball would have to be at least $h+1$. More generally, if the load is at least $h+\xi$ for a positive integer ξ, then all nodes at the bottom level of the truncated history tree of height h that are not black nodes (and there is at least one such node) must be white nodes representing balls that belong to bins with load of at least ξ at their insertion time. We redraw these node as boxes to denote the fact that they represent bins of load at least ξ, and we call them "bin nodes".

The Witness Tree

Let $\xi \in \mathbb{N}$ be a fixed integer to be picked later. For $h, k \in \mathbb{N}$, where $h+\xi \leq k$, a *witness tree* $W_k(h)$ is a truncated history tree of height h of a ball in the set $[k]$, and with two types of leaf nodes, black nodes and "bin" nodes. This means that each internal node has two children, and the node labels belong to the set $[k]$. Each black leaf has a label of an internal node that precedes it in BFS order. Bin nodes are unlabelled nodes that represent bins with load of at least ξ. Bin nodes must all be at the furthest level from the root, i.e., at level h, and there is at least one such node in a witness tree. Notice that every witness tree, by definition, is deterministic, and independent of the total number of bins. An example of a witness tree is shown in Figure 1.4.

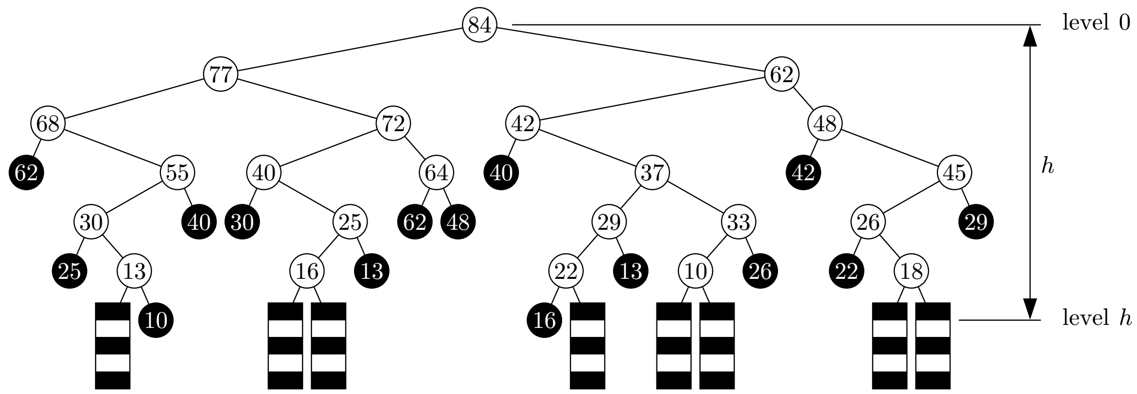

Figure 1.4: A witness tree of height h. The boxes at the lowest level are bin nodes. They represent selected bins with load of at least ξ. The load of the bin that contains ball 84 is at least $h + \xi$.

For any $k, h, d \in \mathbb{N}$, and nonnegative integer z, let $\mathcal{W}_k(h, d, z)$ denote the class of all witness trees $W_k(h)$ that have d internal (white) nodes, and z black nodes (and thus $d - z + 1$ bin nodes). Notice that, by definition, the class $\mathcal{W}_k(h, d, z)$ could be empty, e.g., if $d \notin [h, 2^h)$, $z > d$, or $h > k$. Before we start the proof of Theorem 1.4, we need to establish some facts. First, the number of witness trees in $\mathcal{W}_k(h, d, z)$ can be bound easily.

Lemma 1.1. *For any $k, h, d \in \mathbb{N}$, and integer $z \geq 0$, we have*

$$|\mathcal{W}_k(h, d, z)| \leq 4^d 2^{d+1} d^z k^d .$$

Proof. Without the labelling, there are at most 4^d different shape binary trees, because the shape is determined by the d internal nodes, and hence, the number of trees is the Catalan number $\binom{2d}{d}/(d+1) \leq 4^d$. Having fixed the shape, each of the leaves is of one of two types. Each black leaf can receive one of the d white node labels. Each of the white nodes gets one of k possible labels. $\square$

The next lemma is a simple but crucial fact. We know that in any witness tree $W_k(h) \in \mathcal{W}_k(h,d,z)$, the number of white nodes $d \in [h, 2^h)$ and the number of black nodes $z \in [0, d]$. But can we say more?

Lemma 1.2. *In any witness tree $W_k(h) \in \mathcal{W}_k(h,d,z)$, where $k, h \geq 2$, if the number of white nodes $d \leq 2^{h-\eta}$, where $\eta \geq 1$, then the number of black nodes $z \geq \eta$, i.e.,* $\mathbb{I}_{\left[\,[z\geq\eta]\cup[d>2^{h-\eta}]\,\right]} = 1$.

Proof. Recall that any leaf node is either black or a bin node, every bin node is at distance h from the root, and any witness tree has at least one bin node. Thus, one can see that if we have z black nodes, the number of bin nodes is at least 2^{h-z}. Since $d \leq 2^{h-\eta}$, then $2^{h-\eta} - z + 1 \geq d - z + 1 \geq 2^{h-z}$. If $z = 0$, then we have a contradiction, because $h > 1$. So, assume $z \geq 1$. But then $2^{h-\eta} \geq 2^{h-z}$, that is, $z \geq \eta$. $\square$

Note that, unlike for full or truncated history trees, it is not possible to construct a witness tree $W_k(h)$ for every ball, unless the ball is placed into a bin whose load, just before the insertion, is at least $h + \xi - 1$. Considering algorithm UNIFORM-GREEDYMC(n, k), we say that a witness tree $W_k(h)$ *occurs*, if the random choices of the balls represented by the nodes of the witness tree are exactly as indicated in the witness tree. That is, if we use the information of algorithm UNIFORM-GREEDYMC(n, k), after its termination, to construct a truncated history tree (of height h) for the ball represented by the root of the witness tree, then the history tree must match the witness tree at every level (node for node, color for color, and label for label), except the lowest level where every white node of the truncated history tree must correspond to a bin node in the witness tree and must represent a ball at the top of a bin with at least ξ balls. The bottom line is that a witness tree of height h occurs if and only if a ball is inserted into a bin of load of at least $h+\xi-1$ before its insertion, i.e., the maximum bin load is at least $h+\xi$. We would like to bound the probability that a valid witness tree $W_n(h)$ occurs. Notice that in our case, $k=n$ as the algorithm inserts n balls.

Lemma 1.3. *Considering algorithm* UNIFORM-GREEDYMC(n,n), *we have for any integers* $n,h,d\in\mathbb{N}$, *and integer* $z\in[0,d]$,

$$\sup_{W_n(h)\in\mathcal{W}_n(h,d,z)} \mathbb{P}\left\{W_n(h) \text{ occurs}\right\} \le \frac{1}{\xi^{d-z+1}n^{d+z-1}}\,.$$

Proof. Let $W_n(h)\in\mathcal{W}_n(h,d,z)$ be a fixed witness tree. We use the conditional method to compute the probability that $W_n(h)$ occurs, by looking at each node in BFS order. Suppose that we are at an internal node, say u, in $W_n(h)$. We would like to find the conditional probability that a certain child of node u is exactly as indicated in the witness tree, given that everything is revealed except those nodes that precede u in the BFS order. This depends on the type of the child. If the child is white or black, then the conditional probability is $1/n$, as each ball can be on top of at most one of the n bins which are picked independently and uniformly at random. Multiplying just these probabilities yields $1/n^{d+z-1}$, as there are $d+z-1$ edges in the witness tree that have a white or black nodes as their lower endpoint. If the child is a bin node, however, then the conditional probability is at most $1/\xi$, because there are at most $\lfloor n/\xi \rfloor$ bins with at least ξ balls each is chosen with probability of $1/n$. Since there are $d-z+1$ bin nodes, and the choices are independent, the result follows plainly, by multiplying all the conditional probabilities. □

After these preliminaries, we can now prove the upper bound.

Proof of Theorem 1.4.

Let L_n be the maximum bin load of algorithm Uniform-GreedyMC(n, n). Let $h, \xi, \eta \in [2, \infty)$ be integers to be picked later. By the union bound, we have

$$p \stackrel{\text{def}}{=} \mathbb{P}\{L_n \geq h + \xi\} \leq \mathbb{P}\left\{\bigcup_{W_n(h)} [\, W_n(h) \text{ occurs}\,]\right\} \leq \sum_{W_n(h)} \mathbb{P}\{W_n(h) \text{ occurs}\} \, .$$

Notice that since $h \geq 2$, the number of white (internal) nodes d in any witness tree $W_n(h)$ is at least two, namely, the root and its left child. Using Lemmas 1.1, 1.2 and 1.3, we see that

$$\begin{aligned} p &\leq \sum_{d=2}^{2^h-1} \sum_{z=0}^{d} \sum_{W_n(h) \in \mathcal{W}_n(h,d,z)} \mathbb{P}\{W_n(h) \text{ occurs}\} \\ &\leq \sum_{d=2}^{2^h-1} \sum_{z=0}^{d} |\mathcal{W}_n(h,d,z)| \sup_{W_n(h) \in \mathcal{W}_n(h,d,z)} \mathbb{P}\{W_n(h) \text{ occurs}\} \\ &\leq \sum_{d=2}^{2^h} \sum_{z=0}^{d} \frac{2^{d+1} 4^d d^z n^d}{\xi^{d-z+1}\, n^{d+z-1}} \mathbb{I}_{\left[\,[z \geq \eta] \cup [d > 2^{h-\eta}]\,\right]} \\ &= \frac{2n}{\xi} \sum_{d=2}^{2^h} \left(\frac{8}{\xi}\right)^d \sum_{z=0}^{d} \left(\frac{d\xi}{n}\right)^z \mathbb{I}_{\left[\,[z \geq \eta] \cup [d > 2^{h-\eta}]\,\right]} \, . \end{aligned}$$

Note that we disallow $z = d + 1$, because any witness tree has at least one bin node. We split the sum over $d \leq 2^{h-\eta}$, and $d > 2^{h-\eta}$. For $d \leq 2^{h-\eta}$, we have $z \geq \eta$, and thus

$$\sum_{z=0}^{d} \left(\frac{d\xi}{n}\right)^z \mathbb{I}_{\left[\,[z \geq \eta] \cup [d > 2^{h-\eta}]\,\right]} = \sum_{z=\eta}^{d} \left(\frac{d\xi}{n}\right)^z \leq \left(\frac{d\xi}{n}\right)^\eta \sum_{z=0}^{\infty} \left(\frac{d\xi}{n}\right)^z < 2\left(\frac{d\xi}{n}\right)^\eta ,$$

provided that n is so large that $2^{h+1}\xi \leq n$, (this insures that $d\xi/n < 1/2$). For $d \in (2^{h-\eta}, 2^h]$, we bound trivially, assuming the same large n condition:

$$\sum_{z=0}^{d} \left(\frac{d\xi}{n}\right)^z < 2 \, .$$

In summary, we see that

$$p \leq \frac{4n}{\xi} \sum_{d > 2^{h-\eta}} \left(\frac{8}{\xi}\right)^d + 4\left(\frac{\xi}{n}\right)^{\eta-1} \sum_{d=2}^{2^{h-\eta}} \left(\frac{8}{\xi}\right)^d d^\eta \, .$$

By setting $\xi := 16$, we get

$$p \leq \frac{4n}{\xi 2^{2^{h-\eta}}} + 4C\left(\frac{\xi}{n}\right)^{\eta-1},$$

where $C = \sum_{d\geq 2} d^\eta/2^d$. Clearly, the probability p tends to zero, if we put $\eta := 2$, and $h := \eta + \lceil \log_2 \log_2 n^\eta \rceil$. Notice that ξ and h satisfy the technical condition $\xi 2^{h+1} \leq n$, asymptotically. $\square$

Remark 1.2. The probability p can be made arbitrary small ($p = O(1/n^\delta)$), for any constant $\delta > 0$, by just setting the constant $\eta := \delta + 1$. We have proved that the maximum bin load is at most $\log_2 \log_2 n + 20$, w.h.p. However, by adjusting the constants ξ and η, one can show that the additive constant in the upper bound can be decreased to $12 + \epsilon$, for an arbitrary constant $\epsilon > 0$. Simulation results of algorithm UNIFORM-GREEDYMC(n, n) (see e.g., [13, 170]) show that the additive constant is indeed very small (< 2).

Chapter 2

Nonuniform Two-way Chaining

In this chapter, we analyze the asymptotic worst-case performance of the two-way chaining algorithm NONUNIFORM-SHORTCHAIN(n, m) with two possibly nonuniform hash functions. Roughly speaking, we show that whenever the hashing values behave according to fixed bounded independent probability distributions, the maximum search time is $2\log_2 \log n \pm \Theta(1)$, w.h.p., for $m = \Theta(n)$.

2.1 Motivation

Truly uniform hash functions tend to distribute the keys as evenly as possible over the hash table. This property is also true for conventional (or universal) hash functions which are "almost uniform" as they are chosen randomly from a small set of functions such as the ones in [48, 161, 53]. This means that if the universe set of keys $\mathcal{U}$ is an ordered set, any such hash function is, most likely, not monotonic or order-preserving function. Uniform order-preserving hash functions can be designed, if the key statistics are known priori [155, 76]. If the order-preserving hash function is independent of the key distribution, then the hashed values must be nonuniformly distributed over the hash table, see, e.g., [82] and [41, p. 2]. Order-preserving hash functions are helpful for operations that require sorted or nearly sorted keys like range search and finding the k-nearest neighbors, see [42] for a wide variety of applications.

Lately, there has been growing interest in hashing-based algorithms for solving the (approximate) k-nearest neighbors problem in high-dimensional spaces, see, e.g., [112, 91, 77]. This is due to the efficiency of hashing as a data structure for implementing similarity search in a wide range of database applications [25, 54, 87, 92, 117, 160]. In these applications, a finite number of objects (e.g., images, documents, DNA sequences) is represented by points in a high-dimensional vector space, (e.g., the d-dimensional cube $[0, 1]^d$), such that objects that have similar features are mapped to points that are close to each other. The finite universe set of keys $\mathcal{U}$ is defined to be these points. Searching for a key (or object) in the hash table means finding or approximating the k-nearest neighbors (or similar objects). The heart of this novel approach is a class of hash functions called locally-sensitive hash functions. A function $f : \mathcal{U} \subseteq [0, 1]^d \to \mathcal{T}$ is a locally-sensitive hash function if and only if for all $x, y \in \mathcal{U}$, we have $|f(x) - f(y)| \leq |\mathcal{U}| \, \|x - y\|$, where $\|\cdot\|$ is some given norm defined on $[0, 1]^d$, for example, Euclidean or ℓ_1 norm. Sometimes these functions are called neighborhood-preserving functions [54], or non-expansive functions [112]. In short, such hash functions are sensitive to the similarity of the keys: they map keys that are close to each other, in some sense, to close chains. So, evidently, locally-sensitive hashing is good for fast retrieval, and for minimizing the number of pages consumed by the hash tables. The hashing values of such functions, however, for the same reason explained above, have nonuniform distributions over the hash table.

It is thus important to analyze the performance of hashing schemes with nonuniform hash functions. The worst case performance of classical hashing with chaining where a set of keys $\mathcal{K} \subseteq \mathcal{U}$ are hashed via a single hash function f was studied by Devroye [41] for nonuniform distributions. He represented the hash table by the unit interval $[0,1]$ partitioned into n equal-sized disjoint subintervals. Thus, the hash function f is assumed to map the universe set of keys $\mathcal{U}$ to the unit interval $[0,1]$. Each key x is hashed to the i-th chain, if $f(x)$ belongs to the i-th subinterval. The random hashing locations $f(x)$, for all $x \in \mathcal{U}$, are assumed to be independent and have a common density function h defined over $[0,1]$. Devroye [41] proved that the expected maximum chain length is still asymptotic to $\log n / \log\log n$, provided that the load factor of the hash table is constant, and the density h is bounded. A tight upper bound is also given for unbounded densities.

This motivates us to study the worst-case performance of the two-way chaining paradigm with nonuniform hash functions. Recall that Vöcking's algorithm Left-ShortChain(n,m) is an example of nonuniform two-way chaining where two special independent nonuniform hash functions are used, combined with the tie-breaking rule Always-Go-Left. The length of the longest chain produced by the algorithm is $0.72... \times \log_2 \log n + m/n \pm \Theta(1)$, w.h.p. (Theorems 0.5 and 0.6). The purpose of this chapter is to analyze the worst-case performance of algorithm Nonuniform-ShortChain(n,m) by using the *fixed density model* which we define in the next section. Recall that this algorithm uses two independent hash functions f and g which could have any probability distributions over the hash table. Each key x is inserted into the shortest chain among the chains $f(x)$ and $g(x)$, breaking ties randomly. Before we state the main results, let us first define the stochastic model upon which we build our analysis.

2.2 The Fixed Density Model

Throughout, we assume that algorithm Nonuniform-ShortChain(n,m), which inserts a set of keys $\mathcal{K} \subseteq \mathcal{U}$ of size $m \in \mathbb{N}$ into $n \in \mathbb{N}$ separate chains, is implemented in the following way. The hash table is associated with the unit interval $[0,1]$ which is partitioned into n disjoint equal-sized subintervals denoted by $I_{n,1}, \ldots, I_{n,n}$ where the subinterval $I_{n,i}$ corresponds to the i-th chain of the hash table. More precisely, $I_{n,1} = [0, 1/n]$, and $I_{n,i} = ((i-1)/n, i/n]$, for $i = 2, \ldots, n$. The hash functions f and g map the universe set of keys $\mathcal{U}$ to the unit interval $[0,1]$, and their hashing values behave according to fixed (possibly different) probability density functions h_f and h_g, respectively, defined over $[0,1]$. Thus, for all $x \in \mathcal{U}$, and $i \in [n]$,

$$\mathbb{P}\{f(x) \in I_{n,i}\} = \int_{I_{n,i}} h_f(u)\, du\,, \quad \text{and} \quad \mathbb{P}\{g(x) \in I_{n,i}\} = \int_{I_{n,i}} h_g(u)\, du\,.$$

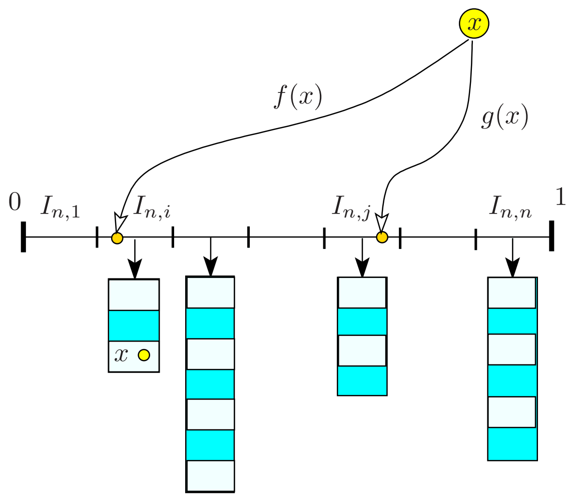

Figure 2.1: Illustration of Nonuniform-ShortChain in the fixed density model. The hash functions f and g map the keys to the unit interval. Key x has two hashing values $f(x) \in I_{n,i}$ and $g(x) \in I_{n,j}$. The key is inserted into the shortest chain.

Notice that the hash functions and their corresponding densities are fixed for all $n \in \mathbb{N}$. All hashing values $f(x)$ and $g(x)$, for all keys $x \in \mathcal{K}$, are assumed to be mutually independent, i.e., each key has two independent hashing values which are also independent from the other keys' hashing values. The keys are inserted on-line and sequentially as follows. For each $x \in \mathcal{K}$, if $f(x) \in I_{n,i}$, and $g(x) \in I_{n,j}$, for some

$i, j \in [n]$, the algorithm inserts the key x into the shortest chain among the i-th and the j-th chains of the hash table, breaking ties randomly. See Figure 2.1.

The main result of this chapter is the following theorem.

Theorem 2.1. *Suppose that algorithm* NONUNIFORM-SHORTCHAIN(n, m), *where* $n, m \in \mathbb{N}$, *is applied in the fixed density model where the hash functions* f *and* g *map the keys according to fixed densities* h_f *and* h_g *over* $[0, 1]$, *respectively. Let* $T_{n,m}$ *be the maximum (successful or unsuccessful) search time. If* $\alpha := m/n = \Omega(1)$, *then* $T_{n,m} \geq 2\max(\alpha, \log_2 \log n - c)$, *w.h.p., for some positive constant* c*; and if* $1/\log n \ll \alpha \ll 1$, *then w.h.p.,* $T_{n,m} \geq (2 - o(1)) \log_2 \log n$. *If both densities are bounded by some constant, then* $T_{n,m} \leq 2\log_2 \log n + O(\alpha)$, *w.h.p. Moreover, if there is a sequence* $\lambda_n = O(\sqrt{\log \log n})$ *such that*

$$\int_{h_f > \lambda_n} h_f(u)\ du\ + \int_{h_g > \lambda_n} h_g(u)\ du = o(1/m)\,,$$

then $T_{n,m} = O((\alpha + 1) \log \log n)$, *w.h.p.*

Other bounds are also presented, including ones on the worst-case search time of the dynamic version of the algorithm. We prove the lower bounds in Section 2.3 by extending the waiting time argument used in the uniform case. In Section 2.4, we apply the witness tree method to prove the upper bound for the case of bounded densities. The case of unbounded densities is treated by using the rejection method. All proofs are presented in the context of the balls-and-bins model, for the sake of simplicity. Formally, we write NONUNIFORM-GREEDYMC(n, m) to denote the greedy multiple-choice allocation process that inserts m balls into n bins where each ball is inserted into the least full bin among two bins chosen according to probability distributions defined on the bins. The allocation process can be implemented in the fixed density model as follows. First of all, we assume that the balls are numbered $1, \ldots, m$ according to their insertion time. Each ball $t \in [m]$, has two independent hashing values X_t and Y_t drawn randomly from the unit interval $[0,1]$ according to the densities h_f and h_g, respectively. Thus, for $i \in [n]$, and $t \in [m]$, we have

$$\mathbb{P}\{X_t \in I_{n,i}\} = \int_{I_{n,i}} h_f(u)\, du\,, \quad \text{and} \quad \mathbb{P}\{Y_t \in I_{n,i}\} = \int_{I_{n,i}} h_g(u)\, du\,.$$

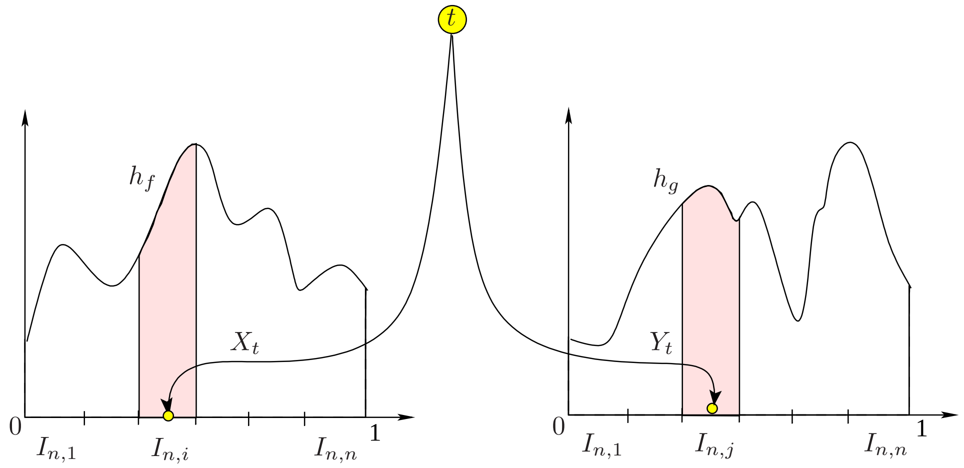

Figure 2.2: Each ball $t \in [m]$ has two hashing values X_t and Y_t drawn from the unit interval according to the densities h_f and h_g, respectively.

The hashing pairs (X_t, Y_t), for all $t \in [m]$, are assumed to be independent. The balls are inserted on-line and sequentially. For each $t \in [m]$, if $X_t \in I_{n,i}$, and $Y_t \in I_{n,j}$, the t-th ball is placed into the least full bin among the i-th and the j-th bin, breaking ties randomly. The maximum bin load of this allocation process is stochastically equivalent to the maximum chain length of algorithm NONUNIFORM-SHORTCHAIN(n, m), when both are implemented in the fixed density model. Hence, we only need to bound the maximum bin load upon termination of the allocation process. Recall that $\alpha = m/n$.

2.3 Lower Bounds

Notice that in the optimal allocation process, each bin receives at least $\lfloor \alpha \rfloor$ and at most $\lceil \alpha \rceil$ balls, and the maximum bin load is equal to $\lceil \alpha \rceil$, deterministically. Thus, to prove the first lower bound stated in Theorem 2.1, we only need to show that the maximum bin load is at least $\log_2 \log n - \Theta(1)$, w.h.p., for all $m = \Omega(n)$.

It is worth mentioning that Vöcking [170], while analyzing the worst-case performance of algorithm LEFT-SHORTCHAIN(n, m), proved that if the bins in algorithm NONUNIFORM-GREEDYMC(n, m), where $m = \Omega(n)$, are chosen according to any arbitrary (possibly dependent) probability distributions, then the maximum bin load—as it is revealed in Theorem 0.5—is at least $0.72... \times \log_2 \log n - \Theta(1)$, w.h.p. Of course, our lower bound is proved only for the fixed density model, but it is obviously better than Vöcking's lower bound by a constant factor.

We begin by proving the following intermediate result for nonuniform distributions that are "sufficiently bounded" in a slightly different model than the fixed density model. Suppose that for each $n \in \mathbb{N}$, we have two sequences of probabilities $p_{n,i}$ and $q_{n,i}$, where $i \in [n]$, according to which the first and the second choices of bins, respectively, are chosen independently. That is, if $(X_t, Y_t) \in [0, 1]^2$ is the hashing pair available for the t-th ball, and $I_{n,i}$ is the subinterval that represents the i-th bin, then for all $t \in [m]$, and $i \in [n]$, we have

$$\mathbb{P}\{X_t \in I_{n,i}\} = p_{n,i}\,, \qquad \text{and} \qquad \mathbb{P}\{Y_t \in I_{n,i}\} = q_{n,i}\,.$$

Of course, $\sum_i p_{n,i} = \sum_i q_{n,i} = 1$, for all $n \in \mathbb{N}$. This model is more general than the fixed density model, because the probabilities $p_{n,i}$ and $q_{n,i}$ could be written as

$$p_{n,i} = \int_{I_{n,i}} h_{1,n}(u)\; du\,, \qquad \text{and} \qquad q_{n,i} = \int_{I_{n,i}} h_{2,n}(u)\; du\,,$$

where $h_{1,n}$ and $h_{2,n}$ are densities over $[0, 1]$ which could be different for each n; while in the fixed density model $h_{1,n} = h_f$, and $h_{2,n} = h_g$, for all $n \in \mathbb{N}$. Nonetheless, the next theorem is true even if the probabilities are obtained from different densities on $[0,1]$ for each n.

Theorem 2.2. *Suppose that algorithm* Nonuniform-GreedyMC(n,m) *is implemented in the model defined above with two sequences of probabilities $p_{n,i}$ and $q_{n,i}$, where $n,m \in \mathbb{N}$. Let $\alpha := m/n$, and $L_{n,m}$ be the maximum bin load upon termination. If there are some constants $\lambda \geq 1$, and $\delta > 0$ such that, for all n large enough,*

$$\sum_{p_{n,i} \leq \lambda/n} p_{n,i} \geq \delta\,, \quad \text{and} \quad \sum_{q_{n,i} \leq \lambda/n} q_{n,i} \geq \delta\,,$$

then $L_{n,m} \geq \log_2 \log_2 n - \max\left(0\,, \lceil 13 - \log_2(c\alpha)\,\rceil\right) - 3$, w.h.p., where $c = \delta^3/(2\lambda - \delta)$.

Proof. Clearly, we can assume that $\alpha > 2^{16}/(c\log_2 n)$, because otherwise, the lower bound is meaningless as it is non-positive. The following proof is a generalization of the waiting time argument of Theorem 1.3. We use the same notation. First, we divide the allocation process into multiple stages. At each stage we refine these survival bins by selecting some of them until we reach the stage where we have only one survival bin at which we stop. The set of the *initial survival bins*, which is denoted here by $\mathcal{I}_n$, has $n_0 > 0$ bins. During any stage in the process, we insert balls sequentially and wait until there are enough survival bins. In the k-th stage, for example, we wait until $n_k < n_{k-1}/2$ bins survive. We shall define the sequence n_k later on. An initial survival bin *survives* the first stage if and only if a ball is inserted into it during the first stage; and for all $k \geq 2$, a survival bin of the $(k-1)$-th stage *survives* the k-th stage if it satisfies one of the following conditions:

1. The bin contains at least k balls before it is chosen by a ball (as one of its two choices) during the k-th stage.

2. The bin contains $k-1$ balls before it is chosen by a ball which is inserted into it during the k-th stage.

A survival bin of the k-th stage has at least k balls in it. In the two conditions above, we say that the ball *helps* the bin to survive. We say that the *survival time* of a survival bin of the k-th stage is t, if the ball that helps it to survive is the t-th ball inserted during the k-th stage. We denote by T_k the number of balls inserted during the k-th stage. This means that T_k is the survival time of the last survival bin of the k-th stage. Our job, then, is to show that one can reach the r-th level, where r is the lower bound we want to prove, by inserting at most m balls, or more formally, $\sum_{k=1}^{r} T_k \le m$, w.h.p., and $n_r \ge 1$. Following the same mathematics, we write $\mathcal{H}_t$ to denote the history up to time t, and A_t to denote the event that the t-th ball inserted during the k-th stage helps a bin to survive. We let S_j be the survival time of the j-th survival bin of the k-th stage. We have seen by using conditional probabilities that if there is a number $\rho_k \in (0,1)$ such that $\mathbb{P}\{A_t \,|\, \mathcal{H}_{t-1}\} > \rho_k$, for all t, then $S_1 \prec E/\rho_k$: S_1 is stochastically smaller than E/ρ_k, where E is an exponential random variable with density e^{-x} on $[0,\infty)$. This means that $\mathbb{P}\{S_1 > t\} \le \mathbb{P}\{E/\rho_k > t\}$, for all t. Thence, we have

$$T_k = S_{n_k} = S_1 + (S_2 - S_1) + \cdots + (S_{n_k} - S_{n_k - 1}) \prec \sum_{j=1}^{n_k} \frac{E_j}{\rho_k} \stackrel{\text{def}}{=} \frac{G_{n_k}}{\rho_k},$$

where $E_1, \ldots, E_{n_k}$ are independent exponential random variables. We also have $\mathbf{E}[\,G_{n_k}/\rho_k\,] = n_k/\rho_k$, and $\mathbf{Var}[\,G_{n_k}/\rho_k\,] = n_k/\rho_k^2$. Consequently, we see that

$$\mathbb{P}\{L_{n,m} < r\} \le \mathbb{P}\left\{\sum_{k=1}^{r} T_k > m\right\} \le \mathbb{P}\{Z_r > m\}, \tag{2.1}$$

where $Z_r := \sum_{k=1}^{r} G_{n_k}/\rho_k$. Thus far, we have followed the same footsteps as the ones in the proof of Theorem 1.3. To complete this proof, we need to define the set of the initial survival bins $\mathcal{I}_n$, and the sequences ρ_k and n_k which are the main differences between the two proofs. Observe that we have not used any thing yet about the probabilities according to which the bins are selected.

Define the following sets: $\mathcal{D}_n := \{i : np_{n,i} \le \lambda\}$, $\mathcal{A}_n := \{i : \delta/2 \le np_{n,i} \le \lambda\}$ and $\mathcal{B}_n := \{i : \delta/2 \le nq_{n,i} \le \lambda\}$. Clearly, for all $i \in \mathcal{A}_n$, the probability that a ball chooses the i-th bin as its first choice is at least $\delta/(2n)$ and at most λ/n. Similarly, $\mathcal{B}_n$ represents the set of all bins that can be chosen by balls as their second choices with probability of at least $\delta/(2n)$ and at most λ/n. Notice that by the assumption, we have $\sum_{i\in\mathcal{D}_n} p_{n,i} \ge \delta$. This yields that

$$\delta \;\le\; \sum_{i\in\mathcal{A}_n} p_{n,i} + \sum_{i\in\mathcal{D}_n-\mathcal{A}_n} p_{n,i} \;<\; |\mathcal{A}_n|\,\frac{\lambda}{n} + (n-|\mathcal{A}_n|)\frac{\delta}{2n} \;=\; |\mathcal{A}_n|\,\frac{2\lambda-\delta}{2n} + \frac{\delta}{2}\,,$$

and hence, $|\mathcal{A}_n| > an$, where $a \stackrel{\text{def}}{=} \delta/(2\lambda-\delta) \in (0,1)$. Similarly, $|\mathcal{B}_n| > an$. There are two cases.

The First Case: $|\mathcal{A}_n \cap \mathcal{B}_n| \ge an/2$

In this case, we define $\mathcal{I}_n$, the set of the initial survival bins, to be the first $\lfloor an/2 \rfloor$ bins in $\mathcal{A}_n \cap \mathcal{B}_n$. Recall that the load of any survival bin of the $(k-1)$-th stage is at least $k-1$. Therefore, if the t-th ball chooses two survival bins of the $(k-1)$-th stage that have not survived the k-th stage yet (and there are at least $n_{k-1} - n_k$ such bins), then the t-th ball helps at least one bin to survive the k-th stage. Clearly, if the i-th bin is a survival bin of the $(k-1)$-th stage, then it must be also an initial survival bin, i.e., $i \in \mathcal{I}_n$. Hence, the probability that it is chosen by a ball as its first choice (or alternatively, as its second choice) is at least $\delta/(2n)$. Since the bins are drawn independently, and $n_k < n_{k-1}/2$, then we have

$$\mathbb{P}\{A_t \,|\, \mathcal{H}_{t-1}\} \ge \left(\frac{\delta(n_{k-1}-n_k)}{2n}\right)^2 > \left(\frac{\delta n_{k-1}}{4n}\right)^2 > 2^{-9}\left(\frac{\delta n_{k-1}}{n}\right)^2 \stackrel{\text{def}}{=} \rho_k\,.$$

Now assume n is sufficiently large, and define the integer sequence

$$n_k = \begin{cases} \lfloor an/2 \rfloor\,, & \text{if } k = 0\,; \\ \left\lfloor an2^{k+\kappa-1} \Big/ 2^{2^{k+\kappa}} \right\rfloor, & \text{for } k \ge 1\,, \end{cases} \tag{2.2}$$

where $\kappa > 1$ is an integer to be picked later. Notice that n_0 is the number of the initial survival bins, and if we define the lower bound we want to prove to be $r := \lceil \log_2 \log_2 n - \kappa - 1 \rceil$, we see that $n_r \geq 1$, for n large enough. The integer κ helps us to satisfy the condition $n_k < n_{k-1}/2$, for all $k \geq 1$, (and so it must be at least 2), and to bound $\mathbf{E}[\,Z_r\,]$, where $Z_r = \sum_{k=1}^{r} G_{n_k}/\rho_k$. Since for all $k > 0$,

$$n_k \leq \frac{an2^{k+\kappa-1}}{2^{2^{k+\kappa}}}\,, \quad \text{and} \quad n_{k-1}^2 \geq \frac{a^2n^2 2^{2(k+\kappa)}}{2^6 \cdot 2^{2^{k+\kappa}}}\,,$$

we have

$$\mathbf{E}[\,Z_r\,] \;=\; \sum_{k=1}^{r} \frac{n_k}{\rho_k} \;=\; \frac{2^9}{\delta^2}\sum_{k=1}^{r}\frac{n_k n^2}{n_{k-1}^2} \;\leq\; \frac{2^{14}}{\delta^2}\sum_{k=1}^{r}\frac{n}{a2^{k+\kappa}} \;\leq\; \frac{2^{14}n}{a\delta^2 2^{\kappa}} \;\leq\; \frac{m}{2}\,,$$

which is true if we set $\kappa = 2 + \max\left(0\,, \lceil 13 - \log_2(a\delta^2\alpha)\,\rceil\right)$. Notice that

$$\kappa = \begin{cases} 2, & \text{if } \alpha \geq 2^{13}/(a\delta^2)\,; \\[2ex] 2 + \lceil 13 - \log_2(a\delta^2\alpha)\,\rceil\,, & \text{otherwise.} \end{cases}$$

Since $n_{r-1}^2 \geq 2^{-8} n (a \log_2 n)^2$, we see that

$$\mathbf{Var}[\,Z_r\,] \;=\; \sum_{k=1}^{r}\frac{n_k}{\rho_k^2} \;\leq\; \rho_r^{-1}\sum_{k=1}^{r}\frac{n_k}{\rho_k} \;=\; 2^9\left(\frac{n}{\delta\, n_{r-1}}\right)^2 \mathbf{E}[\,Z_r\,] \;\leq\; \frac{2^{16} n\, m}{(a\delta\log_2 n)^2}\,.$$

Finally, by returning back to (2.1), and using Chebyshev's inequality, we get

$$\mathbb{P}\left\{L_{n,m} < r\right\} \leq \mathbb{P}\left\{Z_r - \mathbf{E}[\,Z_r\,] > m/2\right\} \leq \frac{\mathbf{Var}[\,Z_r\,]}{(m/2)^2} \leq \frac{2^{18}}{\alpha(a\delta\log_2 n)^2} = o(1)\,,$$

which is true because $\alpha = \Omega(1/\log n)$. This concludes the first case.

The Second Case: $|\mathcal{A}_n \cap \mathcal{B}_n| < an/2$

Since $|\mathcal{A}_n| > an$, then we have $|\mathcal{A}_n - \mathcal{B}_n| = |\mathcal{A}_n| - |\mathcal{A}_n \cap \mathcal{B}_n| > an/2$, and similarly, $|\mathcal{B}_n - \mathcal{A}_n| > an/2$. Let $\mathcal{I}_{n,1}$ be the set of the first $\lfloor \lfloor an/2 \rfloor /2 \rfloor$ bins in $\mathcal{A}_n - \mathcal{B}_n$, and $\mathcal{I}_{n,2}$ be the set of the first $\lceil \lfloor an/2 \rfloor /2 \rceil$ bins in $\mathcal{B}_n - \mathcal{A}_n$. Define $\mathcal{I}_n$, the set of the initial survival bins, to be the union of the two disjoint sets $\mathcal{I}_{n,1}$ and $\mathcal{I}_{n,2}$. Evidently, $|\mathcal{I}_n| = \lfloor an/2 \rfloor$. For simplicity, let us color the bins in $\mathcal{I}_{n,1}$, and $\mathcal{I}_{n,2}$ with white and red, respectively. Observe that if the i-th bin is white, the probability that the first choice of a ball is the i-th bin is at least $\delta/(2n)$; and analogously, if the i-th bin is red, the probability that the second choice of a ball is the i-th bin is also at least $\delta/(2n)$. Since at each stage we have two disjoint sets of survival bins, we require at the k-th stage that exactly $\lfloor n_k/2 \rfloor$ white bins, and $\lceil n_k/2 \rceil$ red bins survive the k-th stage. The total number of survival bins of the k-th stage is still n_k. The load of any survival bin (white or red) of the $(k-1)$-th stage is still at least $k-1$. Let $S_{k,\lfloor n_k/2 \rfloor}$ be the survival time of the last survival white bin of the k-th stage. Similarly, let $S^*_{k,\lceil n_k/2 \rceil}$ be the survival time of the last survival red bin of the k-th stage. Thus, by definition, T_k, which is the survival time of the last survival bin of the k-th stage, can be written as

$$T_k = \max(S_{k,\lfloor n_k/2 \rfloor}, S^*_{k,\lceil n_k/2 \rceil}) < S_{k,\lfloor n_k/2 \rfloor} + S^*_{k,\lceil n_k/2 \rceil} .$$

Let A_t be the event that the t-th ball helps a white bin to survive the k-th stage; and similarly, let A^*_t be the event that the t-th ball helps a red bin to survive the k-th stage. Obviously, if the first choice of the t-th ball is a white bin, the second choice is a red bin, and both choices are survival bins of the $(k-1)$-th stage that have not survived the k-th stage yet, then the ball helps at least one bin to survive. In fact, if one of the bins contains at least k balls, then the ball helps both bins to survive the k-th stage. The worst-case is when both of the chosen bins have load $k-1$; in this case, the white bin, for example, survives with probability $1/2$, because ties are broken randomly. Notice that the number of survival bins of the $(k-1)$-th stage that have not survived the k-th stage yet is at least $\lfloor n_{k-1}/2 \rfloor - \lfloor n_k/2 \rfloor$ white bins plus $\lceil n_{k-1}/2 \rceil - \lceil n_k/2 \rceil$ red bins. Since $n_k < n_{k-1}/2$, and the bins are chosen independently, thence

$$\begin{aligned}\mathbb{P}\{A_t \,|\, \mathcal{H}_{t-1}\} \;&\geq\; \frac{1}{2}\left(\frac{\delta(\lfloor n_{k-1}/2\rfloor - \lfloor n_k/2\rfloor)}{2n}\right)\left(\frac{\delta(\lceil n_{k-1}/2\rceil - \lceil n_k/2\rceil)}{2n}\right)\\ &\geq\; \frac{1}{2}\left(\frac{\delta(n_{k-1}-n_k)}{8n}\right)^2 > \frac{1}{2}\left(\frac{\delta n_{k-1}}{16n}\right)^2 = 2^{-9}\left(\frac{\delta n_{k-1}}{n}\right)^2 = \rho_k\,.\end{aligned}$$

Similarly, $\mathbb{P}\{A_t^* \,|\, \mathcal{H}_{t-1}\} > \rho_k$. Therefore, following the same preliminary argument we started with, we get that both $S_{k,\lfloor n_k/2\rfloor}$ and $S^*_{k,\lceil n_k/2\rceil}$ are stochastically smaller than $G_{\lfloor n_k/2\rfloor}/\rho_k$, and $G_{\lceil n_k/2\rceil}/\rho_k$, respectively, and hence $T_k \prec G_{n_k}/\rho_k$. Thus, the probabilistic duality (2.1) still holds. Since the sequence ρ_k is equal to the one in the first case, and the number of the initial survival bins n_0 is also the same, we can use the same definition of the sequence n_k as in (2.2). The proof now continues exactly as in the previous case to obtain the same lower bound r with the same κ. $\square$

We are almost ready to prove the lower bound for the fixed density model. We now show that the condition of Theorem 2.2 is satisfied in the fixed density model. We write $L_1([0,1])$ to denote the set of all integrable functions on $[0,1]$. We say that a sequences h_n converges to h in $L_1([0,1])$, where $h_n, h \in L_1([0,1])$, to mean that for any Borel set $A \subseteq [0,1]$, we have

$$\lim_{n\to\infty}\int_A |h_n(x) - h(x)|\ dx = 0\,.$$

We say that a sequence h_n converges to h for almost all $x \in [0,1]$ (or almost everywhere on $[0,1]$) to mean that $|h_n(x) - h(x)| \xrightarrow{n} 0$, for all $x \in [0,1]$, except possibly on a set of Lebesgue measure zero. Now we recall the following theorems. For proofs, or more exposure on Lebesgue measure theory and real integration, we recommend [85, 156, 175] and [43, Ch. 2].

Theorem 2.3 (Lebesgue Density Theorem). *Let h be a density on $[0,1]$. Then for almost all $x \in [0,1]$,*

$$\lim_{r\to 0}\frac{1}{r}\int_{B(x,r)} |h(y) - h(x)|\ dy = 0\,,$$

where $B(x,r)$ is a ball centered at x of Lebesgue measure r.

Theorem 2.4 (Scheffé)**.** *If h_n is a sequence of densities on $[0,1]$ that converges almost everywhere to a density h on $[0,1]$, then h_n converges to h in $L_1([0,1])$.*

The general lower bounds stated in Theorem 2.1 follow easily:

Corollary 2.1. *Suppose that algorithm* Nonuniform-GreedyMC(n,m)*, where $n,m \in \mathbb{N}$, is applied in the fixed density model where the hash functions f and g behave according to fixed densities h_f and h_g over $[0,1]$, respectively. Let $L_{n,m}$ be the maximum bin load upon termination. There exists a constant $c > 0$ such that w.h.p.,*

$$L_{n,m} \geq \log_2 \log_2 n - \max\left(0\,, \lceil 13 - \log_2(c\alpha) \rceil\right) - 3\,.$$

Proof. Choose constants $\delta \in (0,1)$, and $\lambda \geq 1$ such that

$$\int_{h_f \leq \lambda} h_f(x)\; dx \geq \delta\,, \quad \text{and} \quad \int_{h_g \leq \lambda} h_g(x)\; dx \geq \delta\,.$$

Notice that these constants depend solely on h_f and h_g, as

$$\lim_{\lambda\to\infty} \int_{h_f \leq \lambda} h_f(x)\; dx = \lim_{\lambda\to\infty} \int_{h_g \leq \lambda} h_g(x)\; dx = 1\,.$$

Recall that the balls select the i-th bin, where $i \in [n]$, according to the probabilities:

$$p_{n,i} := \int_{I_{n,i}} h_f(x)\; dx\,, \quad \text{and} \quad q_{n,i} := \int_{I_{n,i}} h_g(x)\; dx\,.$$

To apply Theorem 2.2, we need to show that there is a constant $\delta' > 0$ such that for n large enough,

$$\sum_{p_{n,i} \leq \lambda/n} p_{n,i} \geq \delta'\,, \quad \text{and} \quad \sum_{q_{n,i} \leq \lambda/n} q_{n,i} \geq \delta'\,.$$

Define the two sets of bins $\mathcal{P}_n = \{i : p_{n,i} \leq \lambda/n\}$, and $\mathcal{Q}_n = \{i : q_{n,i} \leq \lambda/n\}$. Let $h_{f,n}$ and $h_{g,n}$ be the histograms of h_f and h_g, respectively, that is, the following sequences of densities on $[0,1]$:

$$h_{f,n}(x) = \sum_{i=1}^{n} n\, p_{n,i}\; \mathbb{I}_{[x \in I_{n,i}]}\,, \quad \text{and} \quad h_{g,n}(x) = \sum_{i=1}^{n} n\, q_{n,i}\; \mathbb{I}_{[x \in I_{n,i}]}\,.$$

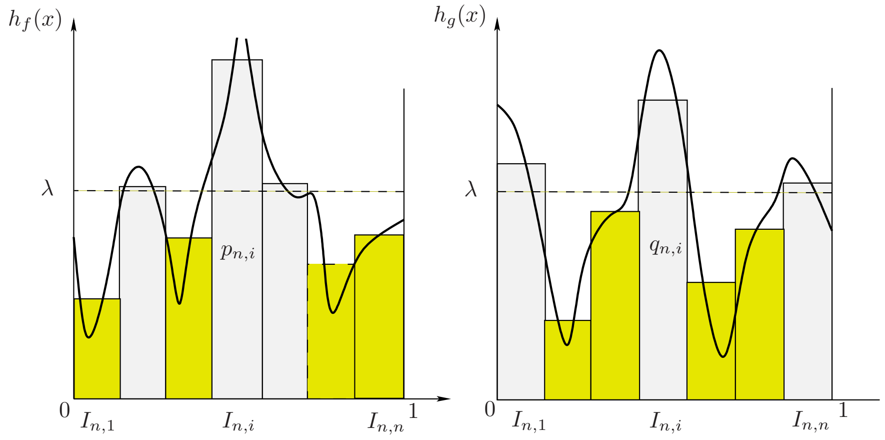

Figure 2.3: The histograms of h_f and h_g. The figures also show the bins (or the subintervals) that are chosen with probabilities of at most λ/n.

The Lebesgue density theorem states that the sequences $h_{f,n}$ and $h_{g,n}$ converge almost everywhere to h_f and h_g, respectively, on $[0,1]$. This is because every subinterval $I_{n,i}$ is a Borel set whose size is $1/n$, and if $x \in [0,1]$, then $x \in I_{n,j}$, for some $j \in [n]$. Hence for almost all $x \in [0,1]$, we have

$$\begin{aligned} |h_{f,n}(x) - h_f(x)| &= \left| n \int_{I_{n,j}} h_f(y)\ dy - n \int_{I_{n,j}} h_f(x)\ dy \right| \\ &\leq n \int_{I_{n,j}} |h_f(y) - h_f(x)|\ dy \xrightarrow{n} 0\,. \end{aligned}$$

Subsequently, Scheffé's theorem implies that $h_{f,n}$ and $h_{g,n}$ converge to h_f and h_g, respectively, in $L_1([0,1])$, which means that

$$\int\limits_{h_{f,n}\leq\lambda} h_{f,n}(x)\ dx \xrightarrow{n} \int\limits_{h_f\leq\lambda} h_f(x)\ dx\,, \quad \text{and} \quad \int\limits_{h_{g,n}\leq\lambda} h_{g,n}(x)\ dx \xrightarrow{n} \int\limits_{h_g\leq\lambda} h_g(x)\ dx\,.$$

Let

$$I_{\mathcal{P}_n} := \bigcup_{i\in\mathcal{P}_n} I_{n,i}\,, \quad \text{and} \quad I_{\mathcal{Q}_n} := \bigcup_{i\in\mathcal{Q}_n} I_{n,i}\,.$$

Since $I_{\mathcal{P}_n} = \{x : h_{f,n}(x) \le \lambda\}$ and $I_{\mathcal{Q}_n} = \{x : h_{g,n}(x) \le \lambda\}$, we have

$$p_n \stackrel{\text{def}}{=} \sum_{i\in\mathcal{P}_n} p_{n,i} = \int_{I_{\mathcal{P}_n}} h_{f,n}(x)\ dx \ \xrightarrow{n} \int_{h_f \le \lambda} h_f(x)\ dx \ \ge \ \delta\,,$$

and similarly,

$$q_n \stackrel{\text{def}}{=} \sum_{i\in\mathcal{Q}_n} q_{n,i} = \int_{I_{\mathcal{Q}_n}} h_{g,n}(x)\ dx \ \xrightarrow{n} \int_{h_g \le \lambda} h_g(x)\ dx \ \ge \ \delta\,.$$

Thence, there exists $\hat{n} \in \mathbb{N}$ such that, for all $n > \hat{n}$, both p_n and q_n are at least $\delta/2$. Therefore, by Theorem 2.2, the result follows with the constant $c = \delta^3/(16\lambda - 4\delta)$. $\square$

2.4 Upper Bounds

In this section, we focus on the second part of Theorem 2.1 about the upper bounds for the worst-case search time of algorithm Nonuniform-ShortChain(n, m). We will bound the maximum bin load of algorithm Nonuniform-GreedyMC(n, m) in the fixed density model. We first deal with bounded densities.

2.4.1 Bounded Densities

A density h over $[0, 1]$ is said to be bounded by a constant λ if and only if $h(x) \le \lambda$, for almost all $x \in [0, 1]$. Notice that $\lambda \ge 1$ because h is a probability density function. We shall establish the following theorem.

Theorem 2.5. *Suppose that algorithm* Nonuniform-GreedyMC(n, m)*, where* $n, m \in \mathbb{N}$*, is applied in the fixed density model where the hash functions* f *and* g *behave according to fixed densities* h_f *and* h_g *over* $[0, 1]$*, respectively. Let* $L_{n,m}$ *be the maximum bin load upon termination. Let* $\alpha := m/n$*. Suppose that both* h_f *and* h_g *are bounded by some constant* λ*. Then* $L_{n,m} \le \log_2 \log_2 n + 16\lambda^2\alpha + 4$*, w.h.p.*

Moreover, for any constant $\epsilon > 1$, there is a constant $c(\epsilon)$ such that if $\alpha \geq c \log n$, then $L_{n,m} < 2\epsilon\lambda\alpha$, w.h.p.

In light of Theorem 2.2, it is clear that Theorem 2.5 has the best asymptotic first term when $1/\log n \ll \alpha \ll \log\log n$, and it is optimal modulo a multiplicative factor for $\alpha = \Omega(\log\log n)$. Recall that Theorem 0.3 reveals that in the case of uniform densities, the maximum bin load is indeed $\log_2 \log n + \alpha \pm \Theta(1)$, w.h.p.

We prove Theorem 2.5 by using the method of witness tree we explained in Section 1.4. The proof, more or less, is the same as the uniform case, except in computing the probability of a witness tree. Recall that we write $W_k(h)$, where $h, k \in \mathbb{N}$, to denote a *witness tree* of height h whose root node represents a ball in the set $[k]$. The internal nodes of any witness tree are white labelled nodes, and the leaves are either black nodes or bin (unlabelled) nodes that represent bins with load of at least ξ, where $\xi \in \mathbb{N}$ is a fixed integer to be picked later on. Clearly, $h + \xi \leq k$. Every white node is labelled with a number of a ball that belongs to the set $[k]$. Each black leaf has a label of an internal node that precedes it in BFS order. Nodes represent balls which are identified with the bins that harbor them. For any $k, h, d \in \mathbb{N}$, and nonnegative integer z, we denote by $\mathcal{W}_k(h, d, z)$ the class of all witness trees $W_k(h)$ that have d white nodes, and z black nodes, (and hence, $d - z + 1$ bin nodes). Since we are dealing with algorithm NONUNIFORM-GREEDYMC(n, m), the existence of a witness tree $W_m(h)$ implies the existence of a bin of load at least $h+\xi$, upon termination. The main argument, then, is to show that a witness tree $W_m(h)$ occurs asymptotically almost surely.

Obviously, Lemmas 1.1 and 1.2 can be applied here as they have nothing to do with probabilities. Thus, we only need to bound the probability that a witness occurs. Notice that since both of the densities are bounded by $\lambda \geq 1$, then for all $i \in [n]$, and

$t \in [m]$, we have

$$\mathbb{P}\{X_t \in I_{n,i}\} \le \lambda/n\,, \quad \text{and} \quad \mathbb{P}\{Y_t \in I_{n,i}\} \le \lambda/n\,,$$

where (X_t, Y_t) is the hashing pair available for the t-th ball, and $I_{n,i}$ is the subinterval that represents the i-th bin. We have the following lemma.

Lemma 2.1. *Let $n, m \in \mathbb{N}$. Let A be the event that upon termination of algorithm* Nonuniform-GreedyMC(n, m), *there are at most $\lfloor n/\xi \rfloor$ bins with at least ξ balls. Then for any integers $n, h, d \in \mathbb{N}$, and integer $z \in [0, d]$, we have*

$$\sup_{W_m(h) \in \mathcal{W}_m(h,d,z)} \mathbb{P}\{W_m(h) \text{ occurs} \,|\, A\} \le \frac{\lambda^{2d}}{\xi^{d-z+1} n^{d+z-1}}\,.$$

Proof. Let $W_m(h) \in \mathcal{W}_m(h, d, z)$ be a fixed witness tree. To compute the probability that $W_m(h)$ occurs, we follow the same technique we used to prove Lemma 1.3 for the uniform case. Using the conditional method, we look at each node in BFS order. Recall that the bins are drawn independently. Suppose that we are at an internal node, say u, in $W_m(h)$. Then, conditioning on A and on everything revealed except those nodes that precede u in the BFS order, the probability that one given child of node u is exactly as indicated in the witness tree depends on the type of the child. If the child is white or black, then the conditional probability is not more than λ/n. Since there are $d + z - 1$ edges in the witness tree that have a white or black node as their lower endpoint, then by multiplying just these probabilities, we have $(\lambda/n)^{d+z-1}$. If the child is a bin node, however, then the conditional probability is at most λ/ξ, because there are at most $\lfloor n/\xi \rfloor$ bins with at least ξ balls (given A is true). Since there are $d-z+1$ bin nodes, this yields $(\lambda/\xi)^{d-z+1}$. The result follows by multiplying the conditional probabilities. □

The next lemma shows that event A in Lemma 2.1 is most likely to be true, for sufficiently large ξ.

Lemma 2.2. *Let $n, m \in \mathbb{N}$. For $\ell \in [n]$, let N_ℓ be the number of bins of load at least ℓ, upon termination of algorithm* NONUNIFORM-GREEDYMC(n,m). *If $\alpha = o(n^{1/3})$, and $6\lambda\alpha \le \ell = O(\alpha)$, then $N_\ell \le \lfloor n/\ell \rfloor$, w.h.p.*

Proof. Fix $\ell \ge 6\lambda\alpha$. For $i \in [n]$, let L_i be the load of the i-th bin after the insertion of m balls. Clearly, $L_i = \sum_{j=1}^{m} U_{ji}$, where U_{ji} is the indicator that ball j is inserted into bin i. The variables U_{ji} are not independent. However, $U_{ji} \le V_{ji}$, where V_{ji} is the indicator that at least one of the choices available for ball j is bin i. Evidently, the variables V_{ji} are independent, and $\mathbb{P}\{V_{ji} = 1\} \le 2\lambda/n$, for all $j \in [m]$. Thus, L_i is stochastically smaller than $\mathrm{Bin}(m, 2\lambda/n)$, for any $i \in [n]$. Therefore, Chebyshev's inequality yields that

$$\mathbf{E}[N_\ell] = \sum_{i=1}^{n} \mathbb{P}\{L_i \ge \ell\} \le n\mathbb{P}\{\mathrm{Bin}(m, 2\lambda/n) \ge \ell\} \le \frac{2\lambda\alpha n}{\ell^2(1-2\lambda\alpha/\ell)^2} \le \frac{3n}{4\ell},$$

because $\ell \ge 6\lambda\alpha$. Observe that N_ℓ can be expressed as a function of $2m$ independent hashing values, and if one of these values is changed, then N_ℓ may decrease or increase by at most one. Thus, by McDiarmid's inequality (Lemma 0.3), we obtain

$$\mathbb{P}\{N_\ell \ge n/\ell\} \le \mathbb{P}\left\{N_\ell - \mathbf{E}[N_\ell] \ge \frac{n}{4\ell}\right\} \le \exp\left(\frac{-n}{16\alpha\ell^2}\right) = o(1),$$

if $\ell = O(\alpha)$, and $\alpha = o(n^{1/3})$. $\square$

Proof of Theorem 2.5.

First of all, the second part of the theorem can be easily seen, if we imagine that the greedy allocation process is modified as follows. Instead of inserting each ball into the least full bin among its two choices, we place it into the first bin and we insert another auxiliary ball into the second one. Then the maximum bin load in this allocation process is greater then $L_{n,m}$. The modified process is equivalent to the nonuniform classical allocation process with $2m$ balls and n bins where the odd-numbered, and the even-numbered balls are inserted into bins chosen independently at random according to the densities h_f, and h_g, respectively. Since the densities are bounded by λ, then any bin is chosen—by odd- or even-numbered balls—with probability of at most λ/n. Thus, the load of any bin in this modified process is stochastically smaller than $\text{Bin}(2m, \lambda/n)$. By Angluin-Valiant's binomial tail inequality, we see that for any constant $\epsilon > 1$,

$$\begin{aligned}\mathbb{P}\left\{L_{n,m} \geq 2\epsilon\lambda\alpha\right\} &\leq n\,\mathbb{P}\left\{\text{Bin}(2m, \lambda/n) > 2\epsilon\lambda\alpha\right\} \\ &\leq n\exp\left(-2(\epsilon\log\epsilon - \epsilon + 1)\lambda\alpha\right) \;=\; o(1)\,,\end{aligned}$$

if $\alpha \geq c\log n$, for some constant $c > (2(\epsilon\log\epsilon - \epsilon + 1)\lambda)^{-1}$. Alternatively, one can prove this also by using the fact that each bin load in the greedy allocation process is stochastically smaller than $\text{Bin}(m, 2\lambda/n)$, as we have shown in the proof of Lemma 2.2.

Regarding the first upper bound of the theorem, we can assume, subsequently, and without lose of generality, that $1 \leq \alpha = O(\log n)$. Let $h, \xi, \eta \in [2, \infty)$ be integers to be defined later. Let A be the event that upon termination of NONUNIFORM-GREEDYMC(n, m), there are at most $\lfloor n/\xi \rfloor$ bins that harbor at least ξ balls. Clearly,

$$\begin{aligned}\mathbb{P}\left\{L_{n,m} \geq h + \xi\right\} &\leq \mathbb{P}\left\{L_{n,m} \geq h + \xi \,|\, A\right\} + \mathbb{P}\left\{A^c\right\} \\ &\stackrel{\text{def}}{=} p + \mathbb{P}\left\{A^c\right\}\,.\end{aligned}$$

We bound the probability p by using the witness tree method. Recall that in any witness tree $W_m(h) \in \mathcal{W}_m(h, d, z)$, the number of white nodes $d \in [2, 2^h)$, and the number of black nodes $z \in [0,d]$. Therefore, using Lemmas 1.1, 1.2, and 2.1, we get

$$
\begin{aligned}
p &\le \mathbb{P}\left\{\cup_{W_m(h)}\left[W_m(h) \text{ occurs}\right] \mid A\right\} \\
&\le \sum_{W_m(h)} \mathbb{P}\left\{W_m(h) \text{ occurs} \mid A\right\} \\
&\le \sum_{d=2}^{2^h-1}\sum_{z=0}^{d} \sum_{W_m(h)\in\mathcal{W}_m(h,d,z)} \mathbb{P}\left\{W_m(h) \text{ occurs} \mid A\right\} \\
&\le \sum_{d=2}^{2^h-1}\sum_{z=0}^{d} |\mathcal{W}_m(h,d,z)| \sup_{W_m(h)\in\mathcal{W}_m(h,d,z)} \mathbb{P}\left\{W_m(h) \text{ occurs} \mid A\right\} \\
&\le \sum_{d=2}^{2^h}\sum_{z=0}^{d} \frac{2^{d+1}4^d d^z m^d \lambda^{2d}}{\xi^{d-z+1}\, n^{d+z-1}} \mathbb{I}_{\left[\,[z\ge\eta]\cup[d>2^{h-\eta}]\,\right]} \\
&= \frac{2n}{\xi}\sum_{d=2}^{2^h}\left(\frac{8\alpha\lambda^2}{\xi}\right)^d \sum_{z=0}^{d}\left(\frac{d\xi}{n}\right)^z \mathbb{I}_{\left[\,[z\ge\eta]\cup[d>2^{h-\eta}]\,\right]}\,.
\end{aligned}
$$

The following computations are similar to the proof of Theorem 1.4. We proceed by splitting the last sum over $d \le 2^{h-\eta}$, and $d > 2^{h-\eta}$. Clearly, when $d \le 2^{h-\eta}$, we have $z \ge \eta$, and thus

$$
\sum_{z=0}^{d}\left(\frac{d\xi}{n}\right)^z \mathbb{I}_{\left[\,[z\ge\eta]\cup[d>2^{h-\eta}]\,\right]} = \sum_{z=\eta}^{d}\left(\frac{d\xi}{n}\right)^z \le \left(\frac{d\xi}{n}\right)^\eta \sum_{z=0}^{\infty}\left(\frac{d\xi}{n}\right)^z < 2\left(\frac{d\xi}{n}\right)^\eta,
$$

provided that n is so large that $2^{h+1}\xi \le n$, (this insures that $d\xi/n < 1/2$). For $d \in (2^{h-\eta}, 2^h]$, we bound trivially, assuming the same large n condition:

$$
\sum_{z=0}^{d}\left(\frac{d\xi}{n}\right)^z < 2\,.
$$

Substituting back, we get

$$
p \le \frac{4n}{\xi}\sum_{d>2^{h-\eta}}\left(\frac{8\alpha\lambda^2}{\xi}\right)^d + 4\left(\frac{\xi}{n}\right)^{\eta-1}\sum_{d=2}^{2^{h-\eta}}\left(\frac{8\alpha\lambda^2}{\xi}\right)^d d^\eta\,. \tag{2.3}
$$

We choose $\xi = \lceil 16\alpha\lambda^2 \rceil$, so that $8\alpha\lambda^2/\xi \le 1/2$. With this choice, we have

$$
p \le \frac{4n}{\xi 2^{2^{h-\eta}}} + 4C\left(\frac{\xi}{n}\right)^{\eta-1},
$$

where $C = \sum_{d\ge 2} d^\eta/2^d$. Clearly, since α is in the range assumed above, the probability p tends to zero, if we put $h = \eta + \lceil \log_2 \log_2 n^\eta \rceil$, and $\eta = 2$. Notice that ξ and h satisfy the technical condition $\xi 2^{h+1} \le n$, asymptotically. Moreover, since ξ satisfies the condition of Lemma 2.2, $\mathbb{P}\{A^c\} = o(1)$. □

Remark 2.1. It is not difficult to see from the last computations that indeed for any constant $\epsilon \in (0,1)$,

$$L_{n,m} \le \log_2 \log_2 n + (8+\epsilon)\alpha\lambda^2 + 3\,,$$

w.h.p. Also, for $\alpha = \Theta(1)$, and any constant $\eta \ge 2$, we have

$$\mathbb{P}\left\{L_{n,m} > \log_2 \log_2 n + \eta + \log_2 \eta + (8+\epsilon)\alpha\lambda^2\right\} = O(n^{1-\eta})\,.$$

The Lightly Loaded Case

In the following theorem, we improve the upper bound for the lightly loaded case, that is, when $\alpha = o(1)$.

Theorem 2.6. *If, in addition to the assumptions of Theorem 2.5, $\alpha \le 1/(16\lambda^2)$, then $L_{n,m} \le \log_2 \log_2 n - \log_2 \log_2 (8\alpha\lambda^2)^{-1} + 4 = \log_2 \log_{(1/\alpha)} n + \Theta(1)$, w.h.p.*

Proof. We adjust the last part of the proof of Theorem 2.5, by noting that setting $\xi = 1$ suffices for our choice of α. Thus, from (2.3) we obtain

$$\begin{aligned} p &\le 4n \sum_{d > 2^{h-\eta}} (8\alpha\lambda^2)^d + \frac{4}{n^{\eta-1}} \sum_{d=2}^{2^{h-\eta}} (8\alpha\lambda^2)^d d^\eta \\ &\le 4n (8\alpha\lambda^2)^{2^{h-\eta}} + \frac{4(8\alpha\lambda^2)^2}{n^{\eta-1}} \sum_{d=0}^{\infty} \frac{(d+2)^\eta}{2^d} \\ &= O(\alpha^2/n)\,, \end{aligned}$$

if we let $\eta = 2$, and

$$h = 2 + \left\lceil \log_2 \left(\frac{\log_2 n^{-2}}{\log_2 (8\alpha\lambda^2)} \right) \right\rceil .$$

Notice that for this analysis to be true, the number of balls m has to be at least $h+\xi$, as it is clear from the definition of the witness tree. Finally, if $m < h + \xi$, then the result is trivially true. $\square$

Theorem 2.6 reveals that if the number of balls $m = \lfloor n^{1-1/2^r} \rfloor$, where

$$0 < r \leq \log_2 \log n - \log_2 \log(4\lambda) - 1 \,,$$

then $L_{n,m} \leq r + \Theta(1)$, w.h.p. This implies that the maximum bin load is constant merely by insuring that the load factor is polynomially smaller than one. Secondly, the greedy multiple-choice allocation process exponentially decreases the maximum bin load all the time. Compare this with the classical allocation process, where each ball is inserted into a bin chosen independently at random, the maximum bin load is at most 2^r, w.h.p., [99, 105]. The third implication of Theorem 2.6 is that increasing the number of bins to $n^{1+o(1)}$ guarantees a plausible decrease in the maximum bin load. Indeed, inserting n balls into $\lfloor n^{1+1/2^r} \rfloor$ bins, where r is as defined above, yields that the maximum bin load is at most $r+\Theta(1)$, w.h.p. For example, we can decrease the maximum bin load to the level $\log\log\log n$, if we increase the bins to $n^{1+1/\log\log n}$. Of course, all the results presented in this section (i,e., for bounded densities) are also true for the uniform allocation process.

The Dynamic Case

In the dynamic case, we are given a sequence of requests $\omega_1, \omega_2, \omega_3, \ldots$, in advance, to be performed by algorithm Nonuniform-GreedyMC in the fixed density model with $n \in \mathbb{N}$ bins, where the j-th request ω_j is either an insertion or a deletion request of a ball. A deletion request asks the algorithm to remove one of the inserted balls from its bin. The ball itself is specified in the request. An insertion request asks the algorithm to insert a new ball. Although the sequence is given in advance, the insertion operations are still performed on-line, without any information about the future requests. The algorithm is applied in the fixed density model: that is, each new ball is placed into the least full bin among two bins X and Y chosen (at insertion time) independently at random according to fixed densities h_f and h_g, respectively, over $[0, 1]$. Suppose that the densities are bounded (say by constant λ), and the sequence of requests is designed such that at any certain time j, there are at most $m \in \mathbb{N}$ balls residing in the bins. Then we claim that the upper bounds of Theorems 2.5 and 2.6 are still true. In particular, the maximum bin load at any time is at most $\log_2 \log n + O(\alpha)$, w.h.p., where $\alpha = m/n$.

The proofs of the above theorems can be adjusted by adjusting the definition of the witness tree. First of all, consider a snapshot of the allocation process at an arbitrary fixed time j, i.e., exactly after the algorithm performs the j-th request. By assumption, there are at most m balls in the bins. For simplicity, assume there are exactly m balls, numbered $1, \ldots, m$ according to their insertion times. Let (X_t, Y_t) be the hashing pair available for ball $t \in [m]$. We identify each ball with the bin that harbors it. The full history tree T_t of a ball $t \in [m]$ (at time j) describes the history of the bin that contains the ball t up to time j; however, we slightly modify its formal definition as follows. The root of T_t is labelled t, and is colored white. The root has two children, a left child corresponding to bin X_t, and a right child corresponding to bin Y_t. The left child is labelled and colored according to the following rules:

(a) If the bin X_t contains some balls whose numbers are less than t (i.e., balls that are inserted before ball t), and the ball with the largest number less than t in that bin, say τ, has not been encountered thus far in the BFS order of the tree T_t, then the left child of t is labelled τ and colored white. Notice that ball τ may not be actually the last ball inserted in bin X_t before insertion time of ball t, but it has existed at that time and still exists in the bin at time j, and no other ball with larger number has this property.

(b) As in case (a), except that τ has already been encountered in the BFS order. We color the node black, and label it τ.

(c) If the bin X_t does not contain any ball with number less than t, then the left child is unlabelled gray node.

Similarly, the right child of t is labelled and colored by following the same rules but with bin Y_t. We continue processing nodes in BFS fashion. A black or gray node in the tree is a leaf and is not processed any further. A white node with label τ is processed in the same way we processed the ball t, but with its two bins X_τ and Y_τ. We continue recursively constructing the tree until all the leaves are black or gray. Obviously, the full history tree has at least one gray node. Notice that the tree is constructed from the snapshot we made at time j, that is, the above rules are applied according to the status of the bins at time j.

The difference between this new definition of the full history tree and the old one is that in the old one, usually the children of any parent node represent the topmost balls in the bins chosen by the parent at insertion time; whereas in the new definition, these topmost balls may not exist at time j, so we have to settle with the balls that have existed at that time and still exist at time j. In the old full history tree, the load of the bin containing the root's ball *is equal to* the length of the shortest path from the root to any gray node. This is not true any more in the new history tree. The load of the bin containing the root's ball at time j *is not less than* the length of the shortest path from the root to any gray node, and this is what we want.

Consequently, based on this new definition of the full history tree, we define the truncated history tree and the witness tree, in the same way as we did before. The truncated history tree of height h of ball t (at time j) is only the top part of the new full history tree T_t which includes all nodes at the first $h+1$ levels, and the remainder is truncated. A witness tree, which we denote by $\widetilde{W}_m(h)$, is a special truncated history tree of height h (at time j) of a ball in the set $[m]$, and with two types of leaf nodes: black nodes and bin nodes which represent bins with load of at least ξ, where ξ is an integer we choose it later on. Bin nodes are at the lowest level. For any $m, h, d \in \mathbb{N}$, and integer $z \geq 0$, we write $\widetilde{\mathcal{W}}_m(h, d, z)$ to denote the class of all witness trees $\widetilde{W}_m(h)$ that have d white nodes, and z black nodes. Observe that a witness tree $\widetilde{W}_m(h)$ exists if and only if the maximum bin load, exactly after algorithm NONUNIFORM-GREEDYMC performs the j-th request, is at least $h+\xi$.

Equivalent versions of Lemmas 1.1, 1.2, 2.1, and 2.2 can be obtained by following the same proofs, because the structure of the new witness tree $\widetilde{W}_m(h)$ is exactly as the old one, there are m balls residing in the bins, and the bins are chosen independently with probability of at most λ/n. Subsequently, the upper bounds stated in Theorems 2.5 and 2.6 can be proved also for the dynamic case. Thus, we get the following theorem. Notice that in the case of uniform densities, the theorem can be combined with remark 1.1.

Theorem 2.7. *Let $\omega_1, \omega_2, \omega_3, \ldots$ be a sequence of insertion and deletion requests to be performed by algorithm* NONUNIFORM-SHORTCHAIN *in a hash table of size n in the fixed density model where the hash functions behave according to fixed bounded densities over $[0,1]$. Suppose that the sequence is specified before the algorithm starts, and it is designed such that at any certain time there are at most $m \in \mathbb{N}$ keys in the table. Let $T_{n,m}$ be the absolute maximum search time. Then $T_{n,m} \leq 2\log_2 \log n + O(m/n)$, w.h.p. Moreover, there is a constant $c > 1$ such that if $\alpha < 1/c$, then $T_{n,m} \leq 2\log_2 \log_{(1/\alpha)} n + \Theta(1)$, w.h.p.*

2.4.2 Unbounded Densities

Suppose that we have $r \in \mathbb{N}$ balls numbered $1, \ldots, r$. Let $\mathcal{H} := \left\{ (\widetilde{X}_t, \widetilde{Y}_t) \;\middle|\; t \in [r] \right\}$ be a set of hashing pairs available for the r balls to be inserted by an algorithm, which we name $\mathbf{A}$, into $n \in \mathbb{N}$ bins by using the greedy multiple-choice paradigm. That is, the balls are inserted sequentially where ball $t \in [r]$ (i.e., the t-th inserted ball) is placed into the least loaded bin among the bins $\widetilde{X}_t$ and $\widetilde{Y}_t$, where ties are broken randomly. Let $\mathcal{J}$ be a proper nonempty subset of $[r]$ of size $m < r$. Let $\mathcal{F} := \left\{ (\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{H} \;\middle|\; t \in \mathcal{J} \right\}$ be a set of hashing pairs available for m balls to be inserted by an algorithm, which we name $\mathbf{B}$, into another set of n bins by following the greedy multiple-choice paradigm. Algorithm $\mathbf{B}$ inserts the balls in the same order as in algorithm $\mathbf{A}$, that is, if $j_1, j_2 \in \mathcal{J}$, and $j_1 < j_2$, then the ball j_1 is inserted before ball j_2. Furthermore, assume that for each $t \in [r]$, we have a uniform $[0, 1]$ random variable R_t that is used by *both* algorithms to break ties: whenever the bins $\widetilde{X}_t$ and $\widetilde{Y}_t$ have the same number of balls, the algorithm inserts ball t into bin $\widetilde{X}_t$, if $R_t < 1/2$, otherwise the ball is inserted into bin $\widetilde{Y}_t$. The bins in both algorithms have the same numbers, and to distinguish between the two sets, let us color the bins used by algorithm $\mathbf{A}$ white, and the bins used by algorithm $\mathbf{B}$ blue.

Lemma 2.3. *Suppose that algorithms* $\mathbf{A}$ *and* $\mathbf{B}$ *are applied, as described above, for some* $n, m, r \in \mathbb{N}$*, where* $m < r$*. Then the maximum bin load of algorithm* $\mathbf{B}$ *is not more than the maximum bin load of algorithm* $\mathbf{A}$*.*

Proof. Let $j_1 < \cdots < j_m$ be the elements of $\mathcal{J}$. For $i \in [n]$, and $t \in [m]$, let $L_A(i, t)$ be the load of the i-th white bin exactly after the insertion of ball j_t by algorithm $\mathbf{A}$. Similarly, let $L_B(i, t)$ be the load of the i-th blue bin immediately after the insertion of ball j_t by algorithm $\mathbf{B}$. Let $L_A(i, 0)$ and $L_B(i, 0)$ be the load of the i-th white and blue bins, respectively, just before the insertion of ball j_1. It is more than enough to prove that $L_B(i, t) \le L_A(i, t)$, for all $i \in [n]$, and $t \in \{0, \ldots, m\}$. We use induction on t. Clearly, when $t = 0$, we have $L_B(i, 0) = 0 \le L_A(i, 0)$, for all $i \in [n]$. Assume that $L_B(i, t) \le L_A(i, t)$ is true, for all $t < T$, and $i \in [n]$. We need to show that $L_B(i, T) \le L_A(i, T)$, for all $i \in [n]$. Without loss of generality, assume that algorithm **B** inserts ball j_T into the blue bin $\widetilde{X}_T$. This means that either $L_B(\widetilde{X}_T, T-1) < L_B(\widetilde{Y}_T, T-1)$, or $L_B(\widetilde{X}_T, T-1) = L_B(\widetilde{Y}_T, T-1)$ and $R_{j_T} < 1/2$. Since only the load of bin $\widetilde{X}_T$ is increased, that is, $L_B(\widetilde{X}_T, T) = L_B(\widetilde{X}_T, T-1) + 1$, thence, for all $i \ne \widetilde{X}_T$, we have

$$L_B(i, T) = L_B(i, T-1) \le L_A(i, T-1) \le L_A(i, T)\,.$$

So we only need to show that $L_B(\widetilde{X}_T, T) \le L_A(\widetilde{X}_T, T)$. If algorithm **A** inserts ball j_T into the white bin $\widetilde{X}_T$, then

$$L_B(\widetilde{X}_T, T) = L_B(\widetilde{X}_T, T-1) + 1 \le L_A(\widetilde{X}_T, T-1) + 1 = L_A(\widetilde{X}_T, T)\,.$$

If algorithm **A** inserts ball j_T into the white bin $\widetilde{Y}_T$, then there are two cases: either $L_A(\widetilde{X}_T, T-1) > L_A(\widetilde{Y}_T, T-1)$, or we have $L_A(\widetilde{X}_T, T-1) = L_A(\widetilde{Y}_T, T-1)$, and $R_{j_T} \ge 1/2$. In the first case, we get

$$\begin{aligned} L_B(\widetilde{X}_T, T) &= L_B(\widetilde{X}_T, T-1) + 1 \le L_B(\widetilde{Y}_T, T-1) + 1 \\ &\le L_A(\widetilde{Y}_T, T-1) + 1 \le L_A(\widetilde{X}_T, T-1) \\ &= L_A(\widetilde{X}_T, T)\,. \end{aligned}$$

If the second case is true, then $L_B(\widetilde{X}_T, T-1) < L_B(\widetilde{Y}_T, T-1)$, otherwise there is a contradiction with $R_{j_T} < 1/2$. This yields that

$$\begin{aligned} L_B(\widetilde{X}_T, T) &= L_B(\widetilde{X}_T, T-1) + 1 \le L_B(\widetilde{Y}_T, T-1) \\ &\le L_A(\widetilde{Y}_T, T-1) = L_A(\widetilde{X}_T, T-1) \\ &= L_A(\widetilde{X}_T, T)\,. \end{aligned}$$

□

Next, we need to recall some properties of the rejection method (see, e.g., [42, Sec. II.3]). It will enable us to make a transition from nonuniform to uniform distributions. For $t \in [m]$, let $(X_t, Y_t) \in [0,1]^2$ be the hashing pair available for the t-th ball, where X_t and Y_t are drawn independently from the densities h_f and h_g, respectively. Assume, temporarily, that both h_f and h_g are bounded by some constant λ. Let $r \in \mathbb{N}$. Suppose that we have a uniform sample of points $(\widetilde{X}_1, U_1), \ldots, (\widetilde{X}_r, U_r)$ and $(\widetilde{Y}_1, V_1), \ldots, (\widetilde{Y}_r, V_r)$, where all the random variables $\widetilde{X}_t, \widetilde{Y}_t, U_t$ and V_t are independent and uniformly distributed over $[0,1]$. Let $\mathcal{F}$ be the set of all points $(\widetilde{X}_t, \widetilde{Y}_t)$, where $t \in [r]$, such that $U_t\lambda \le h_f(\widetilde{X}_t)$, and $V_t\lambda \le h_g(\widetilde{Y}_t)$. Notice that for any $t \in [r]$, the random point $(\widetilde{X}_t, U_t\lambda)$ belongs to the rectangular region $[0,1] \times [0,\lambda]$. This means that by definition,

$$\mathcal{F} := \left\{ (\widetilde{X}_t, \widetilde{Y}_t) : \ (\widetilde{X}_t, U_t\lambda) \in \mathrm{Reg}(h_f), \ \text{and } (\widetilde{Y}_t, V_t\lambda) \in \mathrm{Reg}(h_g) \right\} ,$$

where $\mathrm{Reg}(h_f)$ is the region under the curve h_f in the unit square $[0,1]^2$.

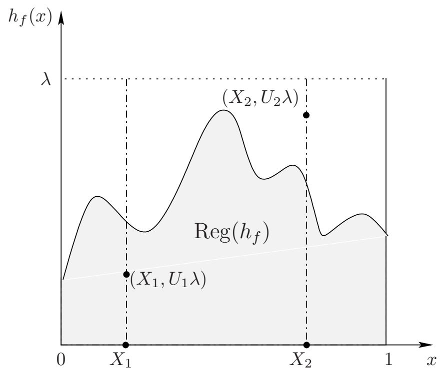

Figure 2.4: The point $(\widetilde{X}_1, U_1\lambda) \in \mathrm{Reg}(h_f)$. The area under the curve is equal to one, because h_f is a denisty.

The following lemma highlights some of the probabilistic properties of the set $\mathcal{F}$.

Lemma 2.4. *The following are true:*

1. *For* $t \in [r]$, *if* $(\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{F}$, *then* $(\widetilde{X}_t, \widetilde{Y}_t) \stackrel{\mathcal{L}}{=} (X_1, Y_1)$, *i.e.*, $\widetilde{X}_t \stackrel{\mathcal{L}}{=} X_1$ *and* $\widetilde{Y}_t \stackrel{\mathcal{L}}{=} Y_1$.

2. *If* $r = 2\lambda^2 m$, *then* $\mathbb{P}\{|\mathcal{F}| \geq m\} \xrightarrow{m} 0$.

Proof. Clearly,

$$\mathbb{P}\left\{U_t\lambda \leq h_f(\widetilde{X}_t)\right\} = \mathbb{P}\left\{(\widetilde{X}_t, U_t\lambda) \in \mathrm{Reg}(h_f)\right\} = 1/\lambda\,.$$

Let $A \subseteq [0,1]$ be any Borel set. Since $\widetilde{X}_t, \widetilde{Y}_t$, and U_t are independent and uniform over $[0,1]$, then we have

$$\begin{aligned}
\mathbb{P}\left\{\widetilde{X}_t \in A \,\middle|\, (\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{F}\right\} &= \mathbb{P}\left\{\widetilde{X}_t \in A \,\middle|\, U_t\lambda \leq f(\widetilde{X}_t)\right\} \\
&= \frac{\mathbb{P}\left\{\left[\widetilde{X}_t \in A\right] \cap \left[U_t \leq f(\widetilde{X}_t)/\lambda\right]\right\}}{\mathbb{P}\left\{U_t\lambda \leq f(\widetilde{X}_t)\right\}} \\
&= \frac{\int_A h_f(x)/\lambda \; dx}{1/\lambda} \;=\; \mathbb{P}\{X_1 \in A\}\,.
\end{aligned}$$

Similarly, $\widetilde{Y}_t \stackrel{\mathcal{L}}{=} Y_1$. It is also evident that $|\mathcal{F}| \stackrel{\mathcal{L}}{=} \mathrm{Bin}(r, 1/\lambda^2)$, because

$$\mathbb{P}\left\{(\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{F}\right\} = 1/\lambda^2\,.$$

Therefore, if $r = 2\lambda^2 m$, then by Chebyshev's inequality, we get

$$\mathbb{P}\{|\mathcal{F}| < m\} \leq \mathbb{P}\{||\mathcal{F}| - \mathbf{E}[|\mathcal{F}|]| \geq m\} \leq \frac{\mathbf{Var}[|\mathcal{F}|]}{m^2} = \frac{2m(1-1/\lambda^2)}{m^2} \xrightarrow{m} 0\,.$$

$\square$

We are now ready to prove the last upper bound in Theorem 2.1. Clearly, if $\alpha = O(1)$, the upper bound is tight up to a multiplicative constant.

Theorem 2.8. *Suppose that algorithm* Nonuniform-GreedyMC(n,m), *where* $n, m \in \mathbb{N}$, *is applied in the fixed density model where the hash functions* f *and* g *behave according to fixed densities* h_f *and* h_g *over* $[0,1]$, *respectively. Let* $L_{n,m}$ *be the* *maximum bin load upon termination. If there is a sequence* $\lambda_n = O(\sqrt{\log\log n})$ *such that*

$$\int\limits_{h_f>\lambda_n} h_f(x)\ dx \ + \int\limits_{h_g>\lambda_n} h_g(x)\ dx = o(1/m)\,, \tag{2.4}$$

then $L_{n,m} = O((\alpha+1)\log\log n)$, *w.h.p.*

Proof. Define the sets $S_n = \{x \mid h_f(x) \le \lambda_n\}$, and $T_n = \{x \mid h_g(x) \le \lambda_n\}$. Let n be large enough such that $\int_{S_n} h_f(x)\ dx > 1/2$, and $\int_{T_n} h_g(x)\ dx > 1/2$, which is obviously possible because of the condition (2.4). Recall that each ball $t \in [m]$ has

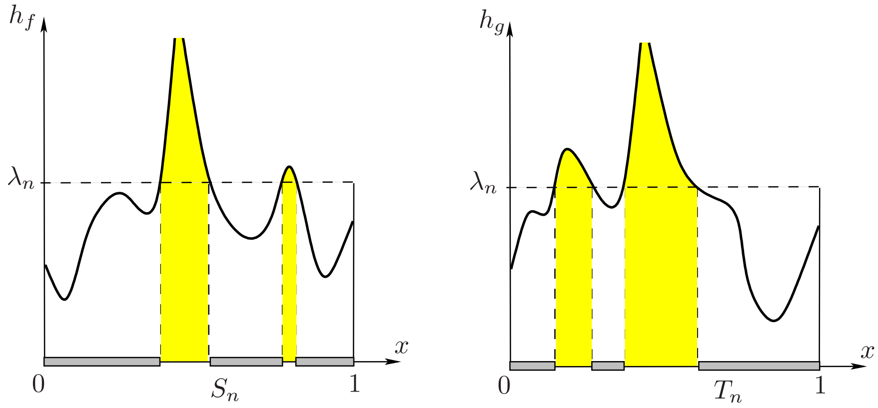

Figure 2.5: The total area of the shaded regions under the curves h_f and h_g should be $o(1/m)$.

a pair of hashing values $(X_t, Y_t) \in [0,1]^2$, where X_t and Y_t are drawn independently from the densities h_f and h_g, respectively. Let D be the event that all the hashing pairs available for the m balls belong to the set $S_n \times T_n$, that is,

$$D = \bigcap_{t=1}^{m} [\, X_t \in S_n, Y_t \in T_n \,]\ .$$

Notice that

$$\begin{aligned}\mathbb{P}\{D^c\} &\le m\,\mathbb{P}\{X_1 \in S_n^c\} + m\,\mathbb{P}\{Y_1 \in T_n^c\} \\ &= m\left(\int\limits_{f>\lambda_n} f(x)\ dx \ + \int\limits_{g>\lambda_n} g(x)\ dx\right) = o(1)\,.\end{aligned}$$

Now we apply the rejection method. Let $r > m$ be an integer to be picked later. Consider the following uniform sample: $(\widetilde{X}_1, U_1), \ldots, (\widetilde{X}_r, U_r)$ and $(\widetilde{Y}_1, V_1), \ldots, (\widetilde{Y}_r, , V_r)$, where all the variables $\widetilde{X}_t, \widetilde{Y}_t, U_t, V_t \in [0, 1]$ are chosen independently and uniformly at random. Let $\mathcal{H} := \Big\{(\widetilde{X}_1, \widetilde{Y}_1), \ldots, (\widetilde{X}_r, \widetilde{Y}_r)\Big\}$ represent a set of uniform hashing pairs available for r balls. We refine the set $\mathcal{H}$ by accepting only those points that have the same distribution as the original hashing pairs (X_t, Y_t). We do that as follows. Let $\mathcal{F}$ be the set of all points $(\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{H}$ such that

$$(\widetilde{X}_t, \widetilde{Y}_t) \in S_n \times T_n,\ (\widetilde{X}_t, U_t\lambda) \in \mathrm{Reg}(h_f),\ \text{and } (\widetilde{Y}_t, V_t\lambda) \in \mathrm{Reg}(h_g)\,.$$

Notice that $(\widetilde{X}_t, U_t\lambda) \in \mathrm{Reg}(h_f)$ if and only if $U_t\lambda_n \le h_f(\widetilde{X}_t)$. Thus, it is not difficult to see, from a generalization of Lemma 2.4, that if the event D is true and $(\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{F}$, then $(\widetilde{X}_t, \widetilde{Y}_t) \stackrel{\mathcal{L}}{=} (X_1, Y_1)$ on $S_n \times T_n$. Since $\int_{S_n} h_f(x)\ dx > 1/2$, and the variables $\widetilde{X}_t, \widetilde{Y}_t, U_t$ and V_t are independent and uniformly distributed over $[0, 1]$, then

$$\mathbb{P}\Big\{\Big[\,U_t\lambda_n \le h_f(\widetilde{X}_t)\Big] \cap \Big[\,\widetilde{X}_t \in S_n\Big]\Big\} = \frac{1}{\lambda_n}\int_{S_n} f(x)\ dx > \frac{1}{2\lambda_n}\,;$$

and similarly,

$$\mathbb{P}\Big\{\Big[\,V_t\lambda_n \le h_g(\widetilde{Y}_t)\Big] \cap \Big[\,\widetilde{Y}_t \in T_n\Big]\Big\} > \frac{1}{2\lambda_n}\,.$$

This means that $\mathbb{P}\Big\{(\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{F}\Big\} > 1/(4\lambda_n^2)$. Hence, $|\mathcal{F}|$ is stochastically greater than $\mathrm{Bin}(r, 1/(4\lambda_n^2))$. By putting $r := \lceil 8\lambda_n^2 m \rceil$, one can see—by using Chebyshev's inequality as in Lemma 2.4—that $\mathbb{P}\{|\mathcal{F}| < m\} = o(1)$. Let $E := [\,|\mathcal{F}| \ge m\,]$. Now suppose that we have an algorithm **A** that inserts r balls into n bins by using the greedy multiple-choice paradigm where the elements of $\mathcal{H}$ are used as hashing pairs for the r balls. That is, each ball $t \in [r]$ is inserted into the least full bin among the two bins $\widetilde{X}_t$ and $\widetilde{Y}_t$, breaking ties randomly. Let M_A be the maximum bin load upon termination of algorithm **A**. Similarly, assuming that E is true, let M_B be the maximum bin load of an algorithm **B** that inserts m balls into another set of n bins by using the greedy multiple-choice paradigm where the first m elements of $\mathcal{F}$ are used as the hashing pairs of the m balls. Furthermore, assume that for each $t \in [r]$, we have a uniform $[0,1]$ random variable R_t that is used by *both* algorithms to break ties: whenever the bins $\widetilde{X}_t$ and $\widetilde{Y}_t$ have the same load, the algorithm inserts ball t into bin $\widetilde{X}_t$, if $R_t < 1/2$, otherwise the ball is inserted into bin $\widetilde{Y}_t$. Observe that given E is true, Lemma 2.3 asserts that $M_B \le M_A$. Since the hashing values of $\mathcal{H}$ are drawn independently from a uniform density over $[0,1]$, and the number of balls $r = \lceil 8\lambda_n^2 m \rceil$, then by Theorem 2.5, we have $M_A \le \zeta_n \stackrel{\text{def}}{=} \log_2 \log_2 n + 128\lambda_n^2 \alpha + 4$, w.h.p. Recall that if the event D is true, and $(\widetilde{X}_t, \widetilde{Y}_t) \in \mathcal{F}$, then $(\widetilde{X}_t, \widetilde{Y}_t) \stackrel{\mathcal{L}}{=} (X_t, Y_t)$. This means that while D and E are true, $L_{n,m} \stackrel{\mathcal{L}}{=} M_B$. Therefore, we conclude that for n large enough,

$$\begin{aligned}
\mathbb{P}\{L_{n,m} > \zeta_n\} &= \mathbb{P}\{[\,L_{n,m} > \zeta_n\,] \cap E \cap D\} + \mathbb{P}\{[\,L_{n,m} > \zeta_n\,] \cap (E \cap D)^c\} \\
&= \mathbb{P}\{[\,M_B > \zeta_n\,] \cap E \cap D\} + \mathbb{P}\{[\,L_{n,m} > \zeta_n\,] \cap (E \cap D)^c\} \\
&\le \mathbb{P}\{[\,M_A > \zeta_n\,] \cap E \cap D\} + \mathbb{P}\{E^c \cup D^c\} \\
&\le \mathbb{P}\{[\,M_A > \zeta_n\,]\} + \mathbb{P}\{E^c\} + \mathbb{P}\{D^c\} = o(1)\,.
\end{aligned}$$

$\square$

Chapter 3

Orientation and Off-line Two-way Chaining

In this chapter, we consider the off-line version of Uniform-ShortChain(n, m). We address the following question. For fixed $k \in \mathbb{N}$, what is the largest $m \in \mathbb{N}$ such that whenever all hashing pairs $(f(x), g(x))$, for $x \in \mathcal{K}$, are known in advance, then asymptotically almost surely, each key $x \in \mathcal{K}$ can be assigned to one of the chains $f(x)$ or $g(x)$ where the maximum chain length is at most k? We model this assignment problem by an orientation of a uniform random graph with n vertices and m edges. We also present some efficient heuristics that find such assignment.

3.1 Motivation

The off-line two-way chaining problem, evidently, provides a useful means for designing efficient static schemes that achieve constant worst-case search time. Such schemes have been widely studied in the literature. Perfect or almost-perfect hashing schemes can be designed in linear time and space, but with hidden large constant factor, not to mention the need for a large number of "good" hash functions. However, by studying off-line two-way chaining, a static hashing scheme whose worst-case search time is at most $2k$ plus 2 (for reading 2 pointers) can easily be designed, once we have an efficient algorithm for finding an assignment for the keys with maximum chain length of at most k. By using techniques like rehashing, such schemes can be modified to support also dynamic data. The off-line analysis of two-way chaining can be also used to measure how good the on-line algorithm is. This is known as *competitive analysis*, which has deep roots in load balancing [14, 12, 13, 39]—another important application of the greedy multiple-choice allocation process.

Pagh [144, 145] studied the off-line version of the cuckoo hashing algorithm which is also the off-line version of the nonuniform two-way chaining algorithm LEFT-SHORTCHAIN. The hash table is partitioned into two disjoint sub-tables $\mathcal{T}_1$ and $\mathcal{T}_2$ of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$. The hash functions f and g are chosen independently and uniformly at random from the sets $\mathbb{F}(\mathcal{U}, \mathcal{T}_1)$, and $\mathbb{F}(\mathcal{U}, \mathcal{T}_2)$, respectively. Pagh showed that w.h.p., there is an assignment of the keys, where each key x is inserted into one of the chains $f(x)$ or $g(x)$, such that maximum chain length is at most 1, (i.e., without any collision), provided that the hash values $(f(x), g(x))$, for all $x \in \mathcal{K}$, are known in advance, and the number of input keys $m = |\mathcal{K}| \leq (1/2 - \epsilon)n$, for some arbitrary constant $\epsilon > 0$. This is also true if the hash functions are chosen from a smaller class of functions with $O(\log n)$ universality, like the ones in [48]. The result is proved by applying the König-Hall theorem [46, Theorem 2.1.2] on a bipartite graph that models the two sub-tables. The assignment of the keys itself can be found by solving a 2-SAT problem that has the following clauses for each pair of distinct keys $x, y \in \mathcal{K}$:

- if $f(x) = f(y)$, we create the clause $(\overline{X}(x) \vee \overline{X}(y))$;
- if $g(x) = g(y)$, we create the clause $(X(x) \vee X(y))$.

where $X(x)$ is a binary variable which is 1 if x should be inserted into $f(x)$, and 0 if it should be inserted into $g(x)$. Since the hash functions are truly uniform, the number of collisions (and hence, the number of clauses) is $O(n)$ in expectation. A linear time algorithm like the one in [64] can be used to find a solution for this instance of the 2-SAT problem. Notice that no polynomial time algorithm is known for solving the general k-SAT problem, for $k \geq 3$.

This chapter, however, is devoted to the off-line version of algorithm UNIFORM-SHORTCHAIN(n, m). To be precise, the word "uniform" should have been added to the title of this chapter. Azar et al. [13] proved that in UNIFORM-SHORTCHAIN(n, n), the input keys can be assigned off-line such that the maximum chain length is at most 10, w.h.p. Their proof is based on modelling the hashing process by an n by n random bipartite graph where the set of left vertices is the set of input keys $\mathcal{K}$, and the set of right vertices is the set of chains $\mathcal{T}$. For each left vertex $x \in \mathcal{K}$, we make two edges $(x, f(x))$ and $(x, g(x))$. The authors showed that there exists an assignment (i.e., a mapping from $\mathcal{K}$ to $\mathcal{T}$) such that the degree of each vertex in $\mathcal{T}$ is at most 10, w.h.p. Indeed, by breaking the set $\mathcal{K}$ into 10 pieces, each of size at most $n/10$, the König-Hall theorem can be used to show that each piece has a perfect matching, w.h.p. Czumaj and Stemann [38] tightened this result and proved that there is an assignment such that the maximum chain length is at most 2, w.h.p. In [39] (the final version of [38]), the authors extended the result for any $m \leq cn$, where $c < 1.67545943...$ is any positive constant. The off-line hashing process is viewed, in the proof of Czumaj and Stemann, as a random graph with n vertices and m multiedges. The assignment problem, then, is transformed into an orientation problem. The goal is to find an orientation such the maximum out-degree is as small as possible. This idea is the core of our study. In the next section, we define this random graph model, we explain in more detail the result of Czumaj and Stemann, and we state our main contributions.

3.2 k-orientability

Recall that algorithm UNIFORM-SHORTCHAIN(n, m) inserts a set of keys $\mathcal{K}$ of size m into a hash table $\mathcal{T} := \{0, \ldots, n-1\}$ consisting of n separate chains, where each key $x \in \mathcal{K}$, has two hashing values $f(x)$ and $g(x)$ which are independently and uniformly distributed over $\mathcal{T}$. Our study is based on the following model.

The Uniform Vertex Model

We represent the off-line process of UNIFORM-SHORTCHAIN(n, m), where $n, m \in \mathbb{N}$, by an undirected random graph denoted by $\mathbb{G}(n, m)$. It has n vertices representing the chains in $\mathcal{T}$, and m multiedges (that may include loops) corresponding to the keys of $\mathcal{K}$. Each edge connects two vertices chosen—one after another—independently and uniformly at random, with replacement, from the set of all n vertices. Finding an assignment of the keys is equivalent to finding an orientation of the graph. Inserting the key x into the chain $f(x)$ means orienting the edge $(f(x), g(x))$ towards the vertex $g(x)$. In the final oriented graph, the *out-degree* of a vertex is defined to be the number of incident edges that are oriented outward. Notice that the out-degree of a vertex u represents the length of chain u. The maximum out-degree of the random graph $\mathbb{G}(n, m)$ is the maximum chain length of the hash table $\mathcal{T}$. For example, if the m edges are realized sequentially, one after another, and each edge (u, v) is oriented, upon realization, toward the second vertex v, which means that the key is always inserted into the first chosen chain u, then the orientation process is equivalent to the on-line algorithm CLASSICCHAIN(n, m), and by Theorem 0.1, the maximum out-degree is asymptotic to $\log n / \log \log n$, in probability, whenever $m = \Theta(n)$. Similarly, if each edge is oriented, upon realization, towards the vertex of minimum out-degree, then Theorems 0.2 and 0.3 say that the maximum out-degree is $\log_2 \log n + m/n \pm \Theta(1)$, w.h.p., for $m = \Omega(n)$, because this orientation method is equivalent to the on-line algorithm UNIFORM-SHORTCHAIN(n, m).

Definition 3.1. An orientation of any graph is called a k-orientation, where $k \in \mathbb{N}$, if and only if the maximum out-degree of the graph is at most k. If a k-orientation exists, we say that the graph is k-orientable.

The k-orientability can be used for other applications such as graph storing and edge membership queries (see [3] for more details). Observe that k-orientability is a decreasing property which means that if $\mathbb{G}(n, m)$ is k-orientable, w.h.p., then subgraphs are also k-orientable w.h.p. For given $n, m \in \mathbb{N}$, we would like to find the smallest integer k such that the random graph $\mathbb{G}(n, m)$ is k-orientable, w.h.p., or equivalently, find the maximum integer $m \leq kn$, for any fixed $k \in \mathbb{N}$, such that $\mathbb{G}(n, m)$ is k-orientable, w.h.p. Throughout, let $c_k = \sup\{c : \mathbb{G}(n, cn) \text{ is } k\text{-orientable w.h.p.}\}$, where $k \in \mathbb{N}$. Our aim is to estimate c_k, the threshold of k-orientability. Obviously, $c_k \leq k$, because $\mathbb{G}(n, kn + 1)$ is not k-orientable, as each vertex can orient outward at most k edges.

Known Results

It is known that for any constant $c \in (0, 1/2]$, the uniform random graph $G(n, cn)$ of Erdös and Rényi [61], which has no loops or multiedges, consists of unicyclic connected components, and isolated trees, and when $c \in (1/2, 1]$, there is also a unique giant connected component of size $\Theta(n)$ that has more than one cycle. This classical result is also true for our random graph $\mathbb{G}(n, m)$, see, e.g., [96]. Clearly, any tree or unicyclic component can be oriented easily such that the maximum out-degree is at most one, see Figure 3.1. A component that has more than one cycle is not 1-orientable. This means that $c_1 = 1/2$.

Czumaj and Stemann [38] proved that w.h.p., the random graph $\mathbb{G}(n, n)$ is 2-orientable, by showing that the giant component can be oriented such that the maxi-

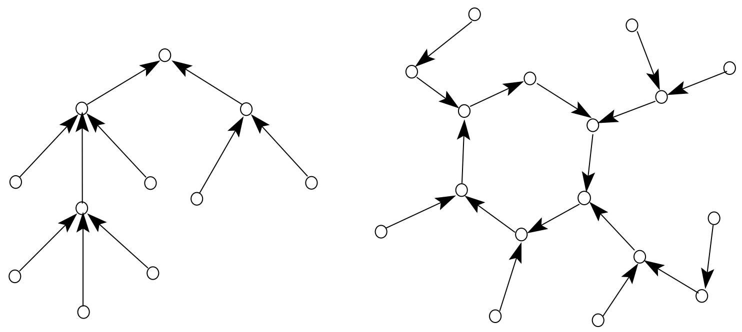

Figure 3.1: Orienting the edges in tree and unicycle components such that the maximum out-degree is at most one. In a tree, a root is fixed first and then all the edges are oriented (in a BFS order) towards the root. In a unicycle, the edges of the cycle are oriented in any direction, and all other edges are oriented towards the cycle.

mum out-degree is at most 2. In [39], Czumaj and Stemann showed that the random graph $\mathbb{G}(n, cn)$ is 2-orientable, w.h.p., for any positive constant $c < 1.67545943...$. The proof uses the threshold for the existence of the 3-core in random graphs [150], where the k-core is the unique maximal subgraph with minimum degree at least k. The result in its general form says that any undirected graph that does not contain a $(k+1)$-core is k-orientable. The idea follows from an algorithm that finds the $(k+1)$-core. The algorithm can be modified to find a k-orientation as follows. The degree of a vertex is defined, here, to be the number of unoriented incident edges.

Greedy-Orient(Graph: G, integer: k)

enqueue all vertices of degree at most k in a queue Q

while $Q \neq \varnothing$ **do**

dequeue a vertex from Q and orient all its edges outward

scan the vertices and enqueue any vertex of degree at most k to the queue Q

end while

So if the graph does not contain a $(k+1)$-core, all the vertices will be enqueued to the queue and hence all the edges will be oriented. Otherwise, the remaining vertices that cannot be enqueued to the queue have degrees of at least $k+1$, and hence they constitute the $(k+1)$-core. The natural question now is: what is the time of the emergence of the $(k+1)$-core? For $k \geq 3$, Pittel, Spencer, and Wormald [150] proved that the random birth time of the k-core in the uniform random graph $G(n,m)$ of Erdös and Rényi is sharply concentrated around $m \approx a_k n/2$, where

$$a_k = \min_{\lambda>0} \frac{\lambda}{\pi_{k-1}(\lambda)}, \quad \text{and} \quad \pi_k(\lambda) = \mathbb{P}\left\{\text{Poisson}(\lambda) \geq k\right\} = \sum_{i=k}^{\infty} \frac{e^{-\lambda}\lambda^i}{i!}.$$

Indeed, they showed that for any $\delta \in (0,1/2)$, if $m \leq a_k n/2 - n^{1-\delta}$, then a.a.s., $G(n,m)$ does not contain any k-core; and if $m \geq a_k n/2 + n^{1-\delta}$, then a.a.s., there is a k-core that is connected and of size $p_k n + o(n)$, where $p_k = \pi_k(\lambda_k)$, and λ_k is the point at which the function $\lambda/\pi_{k-1}(\lambda)$ attains its minimum value. For large k, it is known that $a_k = k + \sqrt{k \log k} + O(\log k)$. This result can be also extended to our model of random graph $\mathbb{G}(n,m)$, where loops and multiedges are allowed. All this shows that $\mathbb{G}(n,m)$ is k-orientable if it does not contain the $(k+1)$-core. However, the converse is not true, i.e., there are graphs that contains the $(k+1)$-core, yet they are still k-orientable, see Figure 3.2. The above analysis only implies the inequality $c_k \geq a_{k+1}/2$, for $k \geq 2$. This means, for instance, that , $c_2 \geq 1.67545943...$, $c_3 \geq 2.57470137...$, and so on, see Table 3.1.

New Results

In Section 3.3, we reprove a result of Frank and Gyárfás [73] that any graph is k-orientable if and only if the number of edges of any subgraph is at most k times the number of its vertices. We use this characterization (in Sections 3.4 and 3.5) to prove that for k sufficiently large, $\rho_k < c_k/k < 1 - \exp\left(-2k\left(1-e^{-2k}\right)\right)$, where

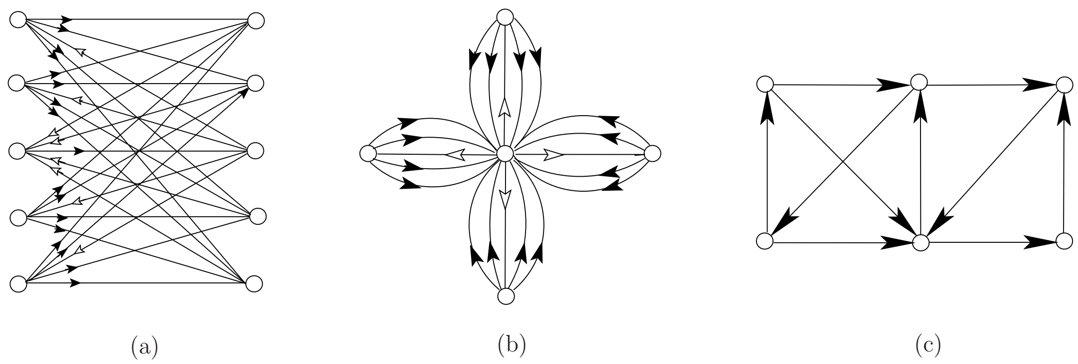

Figure 3.2: The graphs in (a) and (b) are 5-cores, but, clearly, they are 4-orientable. The graph in (c) contains a 3-core, but it is still 2-orientable.

$k+1$	a_{k+1}	$a_{k+1}/2$	p_{k+1}
3	3.35091887...	1.67545943...	0.267580655...
4	5.14940274...	2.57470137...	0.438061712...
5	6.79927548...	3.39963774...	0.538433561...
6	8.36534077...	4.18267038...	0.604638183...
7	9.87529072...	4.93764536...	0.651844404...
8	11.3441289...	5.67206445...	0.687379687...
9	12.7810996...	6.39054984...	0.715208554...
10	14.1923894...	7.09619474...	0.737666503...

Table 3.1: Numerical computations showing the thresholds of the newborn $(k+1)$-core and the ratio of its (giant) size. The threshold $c_k \geq a_{k+1}/2$.

$\rho_k = 1 - 2^k \exp(-k+1+e^{-k/4})$. We also show that for small k, the lower bound on c_k can be computed by solving simultaneously two equations related to the estimation of the upper tail of the binomial distribution. See Tables 3.3 and 3.5 for these computed bounds, for $k \in [2, 55]$, which beat the $(k+1)$-core thresholds except for $k = 1, 2$. Furthermore, we show that w.h.p., a newborn giant subgraph of size at least $\rho_k n$, whose edges are more than k times its vertices, emerges around the time $c_k n$. In other words, we prove the following theorem.

Theorem 3.1. *Let $\mathcal{U}$ be a universe set of keys, and $\mathcal{T}$ be a hash table with $n \in \mathbb{N}$ separate chains. Let $f, g : \mathcal{U} \to \mathcal{T}$ be independent and truly uniform hash functions. For constant $k \in \mathbb{N}$, let m_k be the largest integer such that w.h.p., whenever $\mathcal{K} \subseteq \mathcal{U}$ is a set of keys of size m_k, and the hashing values $f(x)$ and $g(x)$, for $x \in \mathcal{K}$, are known in advance, then each key $x \in \mathcal{K}$ can be assigned to one of the chains $f(x)$ or $g(x)$ so that the maximum chain length is at most k. Then, for k large enough, we have*

$$1 - 2^k \exp\left(-k + 1 + e^{-k/4}\right) < \frac{m_k}{kn} < 1 - \exp\left(-2k\left(1 - e^{-2k}\right)\right) .$$

3.3 Useful Characterization

Throughout this chapter, we use the following notations and definitions. For any graph G, we write $\mathcal{V}(G)$ to denote the set of its vertices. For any set of vertices $S \subseteq \mathcal{V}(G)$, we write $\mathcal{E}(S)$ to denote the multiset of all edges whose endpoints belong to S. The ***density*** of any set of vertices S is the ratio $|\mathcal{E}(S)| / |S|$. If the density of a set S is strictly greater than k, for a positive integer k, we say that S is a ***k-overloaded set***. The ***maximum density*** $\Psi(G)$ of any graph G is defined by

$$\Psi(G) = \max_{S \subseteq \mathcal{V}(G)} \lceil |\mathcal{E}(S)| / |S| \rceil .$$

That is, $\Psi(G)$ is the smallest integer such that $|\mathcal{E}(S)| \leq \Psi(G)\,|S|$, for all $S \subseteq \mathcal{V}(G)$.

We begin by restating the k-orientability property in terms of the edge distribution in the graph. Obviously, if a vertex has more than k loops, or if $|\mathcal{E}(G)| > k\,|\mathcal{V}(G)|$, then the graph G is not k-orientable, as each vertex can orient outward at most k edges. The following lemma generalizes this idea.

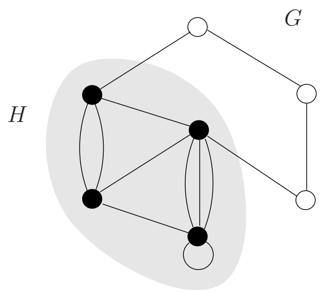

Figure 3.3: The graph G is not 2-orientable, because H is a 2-overloaded set

Lemma 3.1 (Frank and Gyárfás [73]). *Any graph G, possibly containing loops and multiedges, is k-orientable, where $k \in \mathbb{N}$, if and only if its maximum density $\Psi(G) \leq k$, that is, $|\mathcal{E}(S)| \leq k\,|S|$, for all $S \subseteq \mathcal{V}(G)$.*

This means that finding the maximum density of any graph is equivalent to finding the smallest integer k such that the graph is k-orientable. The lemma was originally proved in a more general form by Frank and Gyárfás [73], see also [71, 72]. We have also learned that around the same time D. Avis and B. Reed independently proved the same result in unpublished work. Another proof that uses the König-Hall theorem [46, Theorem 2.1.2] appeared in [3]. However, we give here a new constructive proof based on the following algorithm which for any given graph G, and $k \in \mathbb{N}$, finds either a k-orientation, or a k-overloaded set. Recall that the out-degree of any vertex is the number of incident edges directed outward where a directed loop is counted as one out-directed edge.

ORIENT(graph: G, integer: k)

color all vertices white

for each vertex $u \in \mathcal{V}(G)$ **do**

if u has more than k loops, **then** output $\{u\}$ "is a k-overloaded set" and stop

orient outward up to k edges (if possible) incident to u, giving priority to loops

if out-degree(u) $< k$, **then** color it black

end for

let L be the list of all unoriented edges in G

while $L \neq \varnothing$ **do**

remove an edge (u, v) from the list L, and let $S \leftarrow \varnothing$

using BFS, find an outward directed path from u (or, if not, then from v) to one of the closest black vertices, and while doing so let S be the set of all vertices visited by the two trips of the BFS started from u and v

if such path does not exist, **then** output S "is a k-overloaded set" and stop

if the out-degree of the black vertex the BFS found is $k - 1$, **then** color it white

reorient the path backward toward u (or, respectively, v), and orient the edge (u, v) toward v (or, respectively, u)

end while

Algorithm ORIENT is divided into two phases. The objective of phase 1 (the for loop) is to orient as many edges as possible while keeping the maximum out-degree at most k. This is done by allowing each vertex to choose up to k unoriented edges incident to it and orient them outward, where, of course, the loops have to be oriented first. If there is a vertex with more than k loops, then obviously it is a k-overloaded set, and hence the graph is not k-orientable. The algorithm also keeps track of all vertices whose out-degrees are strictly less than k by coloring them black. Unlike the white vertices whose out-degrees are exactly k, these black vertices have the potential

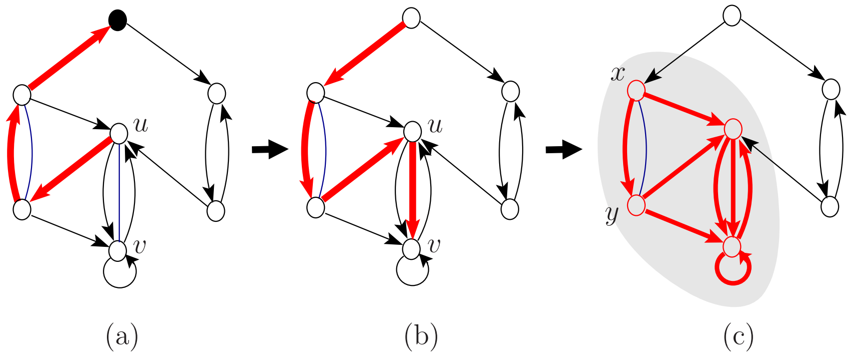

Figure 3.4: (a) The algorithm finds a directed path from u to a black vertex. (b) The algorithm reverses the direction of the path and orients the edge (u, v) toward v. (c) The algorithm cannot find a directed path from x or y to any black vertex, and so it sets S, the k-overloaded set, to be the set of all vertices visited by the last two BFS.

to orient outward more edges than they already have.

If, by the end of phase 1, there are unoriented edges, the algorithm tries in phase 2 (the while loop) to orient them, if possible, while preserving a proper k-orientation. Here the black vertices are utilized. Notice that a black vertex exists if and only if the number of oriented edges is strictly less than kn. However, assuming we have a k-orientable graph, we know that the number of edges in the graph is at most kn. Therefore, if there is an unoriented edge, say (u, v), then the number of oriented edges is strictly less than kn, and hence there must be a black vertex. Notice that (u, v) cannot be oriented in any direction, because both of the vertices u and v are white, thus far! The black vertex can be used to reorient some of the edges while preserving a proper k-orientation, and eventually decrease the out-degree of one the vertices u or v. The algorithm does that by using a breadth-first search to find a directed path from either u or v to one of the closest black vertices. Then it reverses the direction of the path. By doing so, the out-degree of the black vertex is increased by one, the out-degree of u (or v) is decreased by one, while the out-degrees of all other white vertices along the path stay unchanged. This allows us to orient the edge (u, v) toward v, if the path ends up in u, or toward u, if the path ends up in v. So after each reorientation of these paths, we decrease the number of unoriented edges by one, while keeping the maximum out-degree at most k. However, a directed path from either u or v to any of the black vertices may not exist. In this case the graph is not k-orientable as we now show.

Proof of Lemma 3.1

Suppose that G is not k-orientable. We shall prove that the algorithm finds a k-overloaded set. Clearly, if the algorithm does not find a k-overloaded set, then the graph is k-orientable, because the maximum out-degree is at most k after each step in both phases of the algorithm. So assume the algorithm outputs a set S that it claims to be a k-overloaded set. Obviously, a vertex with more than k loops is a k-overloaded set. So, without loss of generality, assume that the algorithm ends up with unoriented edge, say (u, v), and that the breadth-first search was not able to find a directed path from neither u nor v to any black vertex. Thus, S is the set of all vertices visited by the last two unsuccessful breadth-first searches (which started from u and v) before the algorithm halted. In other words, S is just the set of all vertices that can be reached via directed path from u or v, i.e., $u, v \in S$, and if $x \in S$, and (x, y) is an edge oriented toward y, then $y \in S$. Notice that every vertex in S is a white vertex whose out-degree is k, and all its k out-directed incident edges are oriented toward vertices inside S. Hence, the total number of oriented edges that belong to $\mathcal{E}(S)$ is $k\,|S|$. But, the unoriented edge $(u, v) \in \mathcal{E}(S)$. Thus, $|\mathcal{E}(S)| \geq k\,|S| + 1$, i.e., S is a k-overloaded set. The other direction of the lemma is trivial, because in a k-orientation, each vertex can orient outward at most k edges, and hence, $|\mathcal{E}(S)| \leq k\,|S|$, for all $S \subseteq \mathcal{V}(G)$. $\square$

Notice that the worst-case running time of the algorithm ORIENT is $O(n^2)$, if $n = \mathcal{V}(G)$ and the density of G is constant. It is worth mentioning, however, that Aichholzer, Aurenhammer, and Rote [3] gave an $O(n^{3/2})$ worst-case running time algorithm that is based on Hopcroft and Karp's algorithm [89] for computing a maximum matching in a bipartite graph. The authors also presented a linear time heuristic for finding a $2k$-orientation which we call AAR-HEURISTIC.

AAR-HEURISTIC(graph: G, integer: k)

  Let $S \leftarrow \mathcal{V}(G)$

  while $S \neq \varnothing$ **do**

    let u be the vertex with the least degree in S

    if degree$(u) > 2k$, **then** output S "is a $2k$-overloaded set" and stop

    orient outward all edges incident to u, and remove it from S

  end while

It is not difficult to see that if graph G is k-orientable, then any subgraph of G has at least one vertex of degree (i.e., the number of its unoriented incident edges) at most $2k$. Thus, algorithm AAR-HEURISTIC finds a $2k$-orientation if the graph is k-orientable. Of course, to determine the optimal k, one can do an exponential search in $O(\log k)$ steps.

We use Lemma 3.1 to prove the upper and lower bounds on the threshold c_k, in the next sections. Recall that $c_k = \sup\{c : \mathbb{G}(n, cn) \text{ is } k\text{-orientable w.h.p.}\}$. Notice that the existence of a k-overloaded set is an increasing property, i.e., if $\mathbb{G}(n, m)$ contains a k-overloaded set, w.h.p., then for all $m' > m$, the random graph $\mathbb{G}(n, m')$ also contains a k-overloaded set, w.h.p.

3.4 Upper Bounds

Notice that the random graph $\mathbb{G}(n, m)$ is constructed by choosing $2m$ vertices independently and uniformly at random, with replacement, where each two consecutive vertices represent an undirected edge. This means that each loop is chosen with probability $1/n^2$, and each undirected non-loop edge is chosen with probability $2/n^2$. For the remainder of this chapter, we define the ***degree of a vertex*** in the random graph $\mathbb{G}(n, m)$ to be the number of its non-loop incident edges plus twice the number of its loops, i.e., it is the number of times the vertex is chosen during the $2m$ trials of drawing the vertices. Clearly, the degree of any vertex is distributed as $\mathrm{Bin}(2m, 1/n)$. The next theorem bounds c_k from above.

Theorem 3.2. *For any constant integer $k \geq 2$, let γ_k be the unique positive solution of $1 - \gamma - e^{-2\gamma k} = 0$ on $(0, 1)$. Then the threshold $c_k \leq \gamma_k k < \left(1 - e^{-2k(1-e^{-2k})}\right) k$.*

Proof. Suppose that $m = \gamma k n$, for some constant $\gamma \in (0, 1)$. To prove that $c_k < \gamma k$, it suffices to show that the random graph $\mathbb{G}(n, m)$ contains a k-overloaded set, w.h.p. Let S be the set of all non-isolated vertices, i.e., with degree of at least one, in the random graph $\mathbb{G}(n, m)$. Let X be the number of isolated vertices in the random graph $\mathbb{G}(n, m)$, and observe that

$$\begin{aligned} \mathbf{E}[X] &= n\,\mathbb{P}\{\mathrm{Bin}(2m, 1/n) = 0\} = n\left(1 - \frac{1}{n}\right)^{2m} \\ &\geq ne^{-2\gamma k}\left(1 - \frac{1}{n}\right)^{2\gamma k} \geq ne^{-2\gamma k} - 2k\,, \end{aligned}$$

where we have used the inequalities

$$\log(1 - x) \geq -x/(1 - x)\,, \quad \text{and} \quad (1 - x)^{2k} \geq 1 - 2xk\,, \quad \text{valid for } x > 0\,.$$

Notice that $|\mathcal{E}(S)| = m$, and $|S| = n - X$. Moreover, X can be expressed as a function of the $2m$ chosen vertices which are independent, and if one of the vertices is changed, X may increase or decrease by at most one. Therefore, by McDiarmid's inequality (Lemma 0.3), we see that S is a k-overloaded set when γ is large enough. Indeed, for sufficiently large n, we have

$$\begin{aligned}\mathbb{P}\{|\mathcal{E}(S)| \le k\,|S|\} &= \mathbb{P}\{X \le (1-\gamma)n\} \\ &\le \mathbb{P}\left\{X - \mathbf{E}[\,X\,] \le \left(1-\gamma-e^{-2\gamma k}\right)n + 2k\right\} \\ &\le \exp\left(-\left(1-\gamma-e^{-2\gamma k}\right)^2 n/(\gamma k) + 1\right) = o(1)\,,\end{aligned}$$

which is true whenever $f_k(\gamma) \stackrel{\text{def}}{=} 1-\gamma-e^{-2\gamma k} < 0$. In particular, if $\gamma = 1-e^{-2k(1-e^{-2k})}$, then

$$f_k(\gamma) < e^{-2k}\left(e^{2ke^{-2k}} - e^{2ke^{-2k}}\right) = 0\,.$$

This implies that $c_k \le \inf\{\gamma \in (0,1) : f_k(\gamma) < 0\}$. However, $f_k(0) = 0$, $f_k(1/2) > 0$, $f_k(1) < 0$, and since $f_k''(\gamma) = -4\gamma^2 e^{-2\gamma k} < 0$, then f is concave on $[0,1]$. This means that in fact $\gamma_k = \inf\{\gamma \in (0,1) : f_k(\gamma) < 0\}$. $\square$

Remark 3.1. Notice that the upper bound on c_k is obtained by estimating the random time at which the 1-core becomes a k-overloaded subgraph, that is, when the density of the 1-core exceeds k. One can improve this bound by considering instead the density of the $(k+1)$-core. That is, if C_k is the smallest constant c such that, w.h.p., the density of the $(k+1)$-core of the random graph $\mathbb{G}(n, cn)$ is more than k, then $c_k \le C_k$. We know from the work of Pittel, Spencer, and Wormald [150] that for $k \ge 2$, the $(k+1)$-core of the uniform random graph $G(n,m)$, where no loops or multiedges are allowed, emerges around the time $m \approx a_{k+1}n/2$, where

$$a_{k+1} = \min_{\lambda>0} \frac{\lambda}{\pi_k(\lambda)}\,, \quad \text{and} \quad \pi_k(\lambda) = \mathbb{P}\{\text{Poisson}(\lambda) \ge k\}\,.$$

Moreover, for any given constant $c > a_{k+1}/2$, the number of vertices in the $(k+1)$-core of the random graph $G(n, cn)$ is $\pi_{k+1}(\lambda_k(c))n + o(n)$, w.h.p., where $\lambda_k(c)$ is the largest root of the equation $2c = \lambda/\pi_k(\lambda)$. On the other hand, Fountoulakis [70] proved that the number of edges in the $(k+1)$-core of the random graph $G(n, cn)$ is $\lambda_k^2(c)n/(4c) + o(n)$, w.h.p. These results are also expected to be true in the model $\mathbb{G}(n, m)$ which is highly unlikely to have more than a constant number of loops or multiedges. Thus,

$$C_k = \inf \left\{ c > \frac{a_{k+1}}{2} \;\middle|\; \frac{\lambda_k^2(c)}{4c\, \pi_{k+1}(\lambda_k(c))} > k \right\},$$

The following table shows some of the computed values of C_k compared to the upper bound of Theorem 3.2.

k	C_k	$\gamma_k k$
2	1.79402374...	1.960345197...
3	2.87746281...	2.992450613...
4	3.92147910...	3.998654534...
5	4.94775681...	4.999772897...
6	5.96443625...	5.999963132...
7	6.97541865...	6.999994180...
8	7.98282627...	7.999999100...
9	8.98790713...	8.999999863...
10	9.99143452...	9.999999979...

Table 3.2: The threshold $c_k \le C_k \le \gamma_k k$.

3.5 Lower Bounds

We already know that c_k is at least the threshold of the $(k+1)$-core, which is asymptotic to $k/2$ [150]. In this section, we improve this lower bound, and show that indeed $c_k \sim k$, as $k \to \infty$. Observe that that for any set of vertices $S \subseteq \mathcal{V}(\mathbb{G}(n,m))$ of size $i \in \mathbb{N}$, the probability that we choose an edge whose both endpoints belong to S is i^2/n^2, because each vertex is drawn independently and uniformly at random, with replacement. Therefore, $|\mathcal{E}(S)| \stackrel{\mathcal{L}}{=} \mathrm{Bin}(m, i^2/n^2)$. Thence, by Lemma 3.1, the probability that the random graph $\mathbb{G}(n,m)$ is not k-orientable is no more than

$$\sum_{i=1}^{\lfloor m/k \rfloor} \sum_{S:|S|=i} \mathbb{P}\left\{|\mathcal{E}(S)| > ki\right\} \leq \sum_{i=1}^{\lfloor m/k \rfloor} \binom{n}{i} \mathbb{P}\left\{\mathrm{Bin}(m, i^2/n^2) > ki\right\} .$$

We would like to find the maximum m such that the above probability tends to zero as n approaches infinity. The following lemma shows that c_k is at least $k/\sqrt{e}$, asymptotically. However, the approximations used in the proof are not tight enough. Nonetheless, the lemma is an important step towards the main result. Notice that for $i = 0, \ldots, n$, we have

$$\binom{n}{i} \leq \frac{n^i}{i!} \leq \left(\frac{en}{i}\right)^i . \tag{3.1}$$

Lemma 3.2. *Let $k \geq 2$ be any constant integer. The random graph $\mathbb{G}(n, kn)$ does not contain any k-overloaded set of size less than or equal to $ne^{-(k+1)/(k-1)}$, w.h.p. Furthermore, the threshold c_k is at least $ke^{-(k+1)/(2k-1)}$.*

Proof. Let $j = \left\lfloor ne^{-(k+1)/(k-1)} \right\rfloor$. Using (3.1), and inequality (2) in Lemma 0.1, we see that for n large enough, the probability of existence of a k-overloaded set of size of at most j in the random graph $\mathbb{G}(n, kn)$ is not more than

$$\begin{aligned}
\sum_{i=1}^{j} \sum_{|S|=i} \mathbb{P}\left\{|\mathcal{E}(S)| > ki\right\} &\leq \sum_{i=1}^{j} \binom{n}{i} \mathbb{P}\left\{\operatorname{Bin}(kn, i^2/n^2) > ki\right\} \\
&\leq \sum_{i=1}^{j} \left(\frac{en}{i}\right)^i \left(\frac{ei}{n}\right)^{ki} e^{-ki^2/n} \\
&\leq \sum_{i=1}^{j} \left(e^{k+1} (i/n)^{k-1}\right)^i e^{-ki^2/n} \\
&\leq \sum_{i=1}^{\lfloor \log n \rfloor} \frac{e^{k+1}\, i}{n} + \sum_{\lceil \log n \rceil}^{\lfloor j/e \rfloor} \left(e^2 (j/n)^{k-1}\right)^i + \sum_{\lceil j/e \rceil}^{j} e^{-ki^2/n} \\
&\leq \frac{e^{k+1} \log^2 n}{n} + \sum_{\lfloor \log n \rfloor}^{\infty} e^{-i} \; + \; n \exp\left(-e^{-2} k j^2/n\right) \\
&\leq o(1) + \Theta(1/n) + o(1) = o(1)\,.
\end{aligned}$$

Now if $m = \lfloor a_k kn \rfloor$, where $a_k = e^{-(k+1)/(2k-1)}$, then w.h.p., the random graph $\mathbb{G}(n, m)$ is k-orientable, because the probability that there is a k-overloaded set of size greater than j is less than

$$\begin{aligned}
\sum_{i=j}^{\lceil m/k \rceil - 1} \left(\frac{en}{i}\right)^i \left(\frac{emi}{kn^2}\right)^{ki} e^{-mi^2/n^2} &\leq \sum_{i=j}^{\lfloor a_k n \rfloor} \left(e^{k+1} (i/n)^{k-1} a_k^k\right)^i e^{-mi^2/n^2} \\
&\leq \sum_{i=j}^{\lfloor a_k n \rfloor} \left(e^{k+1} a_k^{2k-1}\right)^i e^{-mi^2/n^2} \\
&= \sum_{i=j}^{\lfloor a_k n \rfloor} e^{-mi^2/n^2} \\
&\leq n \exp\left(-mj^2/n^2\right) = o(1)\,.
\end{aligned}$$

$\square$

Lemma 3.2, clearly, improves the lower bound on c_k, for k large enough, but it also says that the size ratio of the smallest k-overloaded set in $\mathbb{G}(n, m)$, where $m \leq kn$, is at least $e^{-(k+1)/(k-1)} \geq e^{-3}$, w.h.p. However, we shall further improve the lower bound on c_k, and see that the size ratio of the newborn k-overloaded set grows exponentially to 1, as $k \to \infty$. We do that by tightening the estimation of the upper tail of the binomial distribution: we shall use inequality (1). For that, we need to define the following positive functions. Fix an integer $k \geq 2$. Suppose that $m = \lfloor \alpha n \rfloor$, for some $\alpha \in (0, k]$. For $p \in (0, \alpha/k)$, let

$$f(k, \alpha, p) = \left(\frac{\alpha(1-p^2)}{\alpha - kp}\right)^{\alpha - kp} \left(\frac{\alpha p}{k}\right)^{kp}, \tag{3.2}$$

and define

$$h(k, \alpha, p) = \begin{cases} 1\,, & \text{if } p = 0\,; \\ p^{-p}(1-p)^{p-1} f(k, \alpha, p)\,, & \text{for } p \in (0, \alpha/k)\,; \\ (\alpha/k)^{2\alpha}\,, & \text{if } p = \alpha/k\,. \end{cases} \tag{3.3}$$

The functions f and h, as we are going to see further on, are strongly related to the function Υ defined in Lemma 0.1. Notice that h is continuous on $[0, \alpha/k]$, and smooth on $(0, \alpha/k)$.

3.5.1 Tight Asymptotic Estimations

Our main asymptotic lower bounds are stated in the following theorem. We use the notation $h_p(k, \alpha, q)$ to denote the partial derivative of h with respect to p evaluated at the point (k, α, q).

Theorem 3.3. *For any fixed integer $k \geq 2$, define*

$$\alpha_k := \sup\left\{\alpha > 0 \,:\, \exists\, \delta \in (0,1) \;\; \textit{such that} \;\; h(k, \alpha, p) \leq \delta, \quad \forall\, p \in (e^{-3}, \alpha/k)\right\},$$

and let $\rho_k := 1 - (2/e)^k e^{1+e^{-k/4}}$. Then the following are true:

1. *The threshold $c_k \geq \alpha_k$; and for k large enough, $\alpha_k > k\rho_k$.*

2. *If s_k is a point at which the function $h(k, \alpha_k, p)$ attains its maximum on the interval $[e^{-3}, \alpha_k/k]$, then $h(k, \alpha_k, s_k) = 1$, and $h_p(k, \alpha_k, s_k) = 0$.*

3. *For fixed $k \geq 2$, and $\alpha \in (\alpha_k, k]$, the equation $h(k, \alpha, p) = 1$ has two positive solutions. Let $q_1(k, \alpha)$ and $q_2(k, \alpha)$ be the smallest and the largest of these solutions. The size ratio of the newborn k-overloaded set is between $q_1(k, c_k)$ and $q_2(k, c_k)$, w.h.p. Moreover, $q_1(k, c_k) > r_k \stackrel{\text{def}}{=} q_1(k, k) \geq \rho_k$, for large k.*

Theorem 3.3 also provides a heuristic for computing the exact value of α_k. Solving the two equations $h(k, \alpha, p) = 1$, and $h_p(k, \alpha, p) = 0$, simultaneously, for any given $k \geq 2$, one can obtain the lower bound α_k. Unfortunately, solving these two equations explicitly is somehow impossible. So we used the mathematical software Maple to solve them numerically. The numerical computations of α_k (see Table 3.3) suggest, indeed, a more tight lower bound on α_k than the one mentioned in the theorem. We conjecture that for all $k \geq 2$,

$$\frac{\alpha_k}{k} \geq 1 - \left(\frac{2}{e}\right)^{k+\sqrt{k}} .$$

Note that this lower bound holds for each computed α_k in Table 3.3. The reader is invited to verify that.

Recall that the k-overloaded set emerges around time $m = c_k n$. Theorem 3.3 reveals that one can lower-bound the size ratio of the newborn k-overloaded set by computing the smallest positive root of $h(k, k, p) = 1$, which we call $r_k = q_1(k, k)$. Obviously, it converges monotonically to one, as k goes to infinity. This is illustrated in Table 3.4, and Figure 3.5. Notice that the newborn k-overloaded set is giant (i.e., of size $\Theta(n)$). This is expected, because it is unlikely that at the beginning of the evolution, a large number of edges land on a very small set. However, while keeping the number of vertices fixed, and as the number of edges m increases away from $c_k n$, the size of the k-overloaded set starts to decrease to one.

k	α_k	s_k	k	α_k	s_k
2	1.30343190...	0.323260552...	29	28.9998095...	0.999858942...
3	2.48312473...	0.533227221...	30	29.9998597...	0.999896314...
4	3.61901095...	0.668655045...	31	30.9998967...	0.999923787...
5	4.71902985...	0.761197567...	32	31.9999239...	0.999943972...
6	5.79256286...	0.826480988...	33	32.9999439...	0.999958807...
7	6.84673418...	0.873351248...	34	33.9999587...	0.999969722...
8	7.88671563...	0.907333583...	35	34.9999696...	0.999977734...
9	8.91625922...	0.932106804...	36	35.9999776...	0.999983633...
10	9.93810345...	0.950220868...	37	36.9999835...	0.999987967...
11	10.9542584...	0.963487239...	38	37.9999878...	0.999991145...
12	11.9662054...	0.973211591...	39	38.9999910...	0.999993491...
13	12.9750390...	0.980342855...	40	39.9999934...	0.999995214...
14	13.9815687...	0.985573824...	41	40.9999951...	0.999996477...
15	14.9863940...	0.989411495...	42	41.9999964...	0.999997413...
16	15.9899586...	0.992227335...	43	42.9999973...	0.999998091...
17	16.9925912...	0.994293662...	44	43.9999980...	0.999998605...
18	17.9945347...	0.995810163...	45	44.9999985...	0.999998966...
19	18.9959692...	0.996923274...	46	45.9999989...	0.999999233...
20	19.9970278...	0.997740402...	47	46.9999992...	0.999999433...
21	20.9978087...	0.998340329...	48	47.9999994...	0.999999583...
22	21.9983847...	0.998780842...	49	48.9999995...	0.999999709...
23	22.9988094...	0.999104346...	50	49.9999996...	0.999999778...
24	23.9991226...	0.999341946...	51	50.9999997...	0.999999844...
25	24.9993535...	0.999516470...	52	51.9999998...	0.999999878...
26	25.9995236...	0.999644679...	53	52.9999998...	0.999999916...
27	26.9996490...	0.999738877...	54	53.9999999...	0.999999939...
28	27.9997414...	0.999808082...	55	54.9999999...	0.999999947...

Table 3.3: The numerical solutions α_k and s_k of the equations $h(k, \alpha, p) = 1$ and $h_p(k, \alpha, p) = 1$. The threshold $c_k \geq \alpha_k$ which is strictly greater than the threshold of the $(k+1)$-core (in Table 3.1), except for α_2 and α_3.

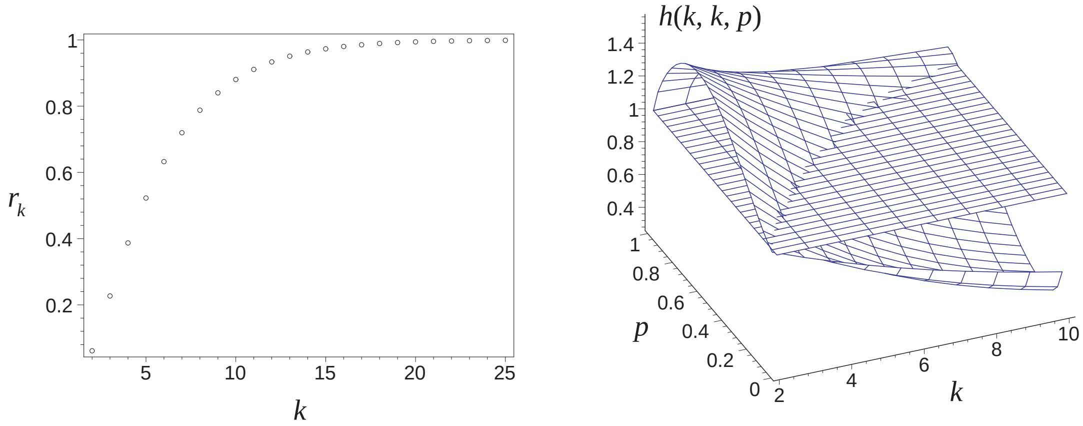

Figure 3.5: The lower bound r_k on the size ratio of the newborn k-overloaded set converges exponentially to one. It is where the curve of $h(k, k, p)$ intersects 1.

k	r_k	k	r_k	k	r_k
2	0.061389845...	11	0.910842703...	20	0.994142089...
3	0.226773619...	12	0.933714444...	21	0.995686702...
4	0.387019206...	13	0.950841371...	22	0.996824664...
5	0.522609724...	14	0.963613455...	23	0.997662747...
6	0.632575890...	15	0.973107193...	24	0.998279818...
7	0.719774099...	16	0.980146405...	25	0.998734074...
8	0.787849394...	17	0.985355668...	26	0.999068429...
9	0.840355794...	18	0.989205077...	27	0.999314504...
10	0.880458374...	19	0.992046478...	28	0.999495594...

Table 3.4: The size ratio of the newborn k-overloaded set is at least r_k, the solution of the equation $h(k, k, p) = 1$.

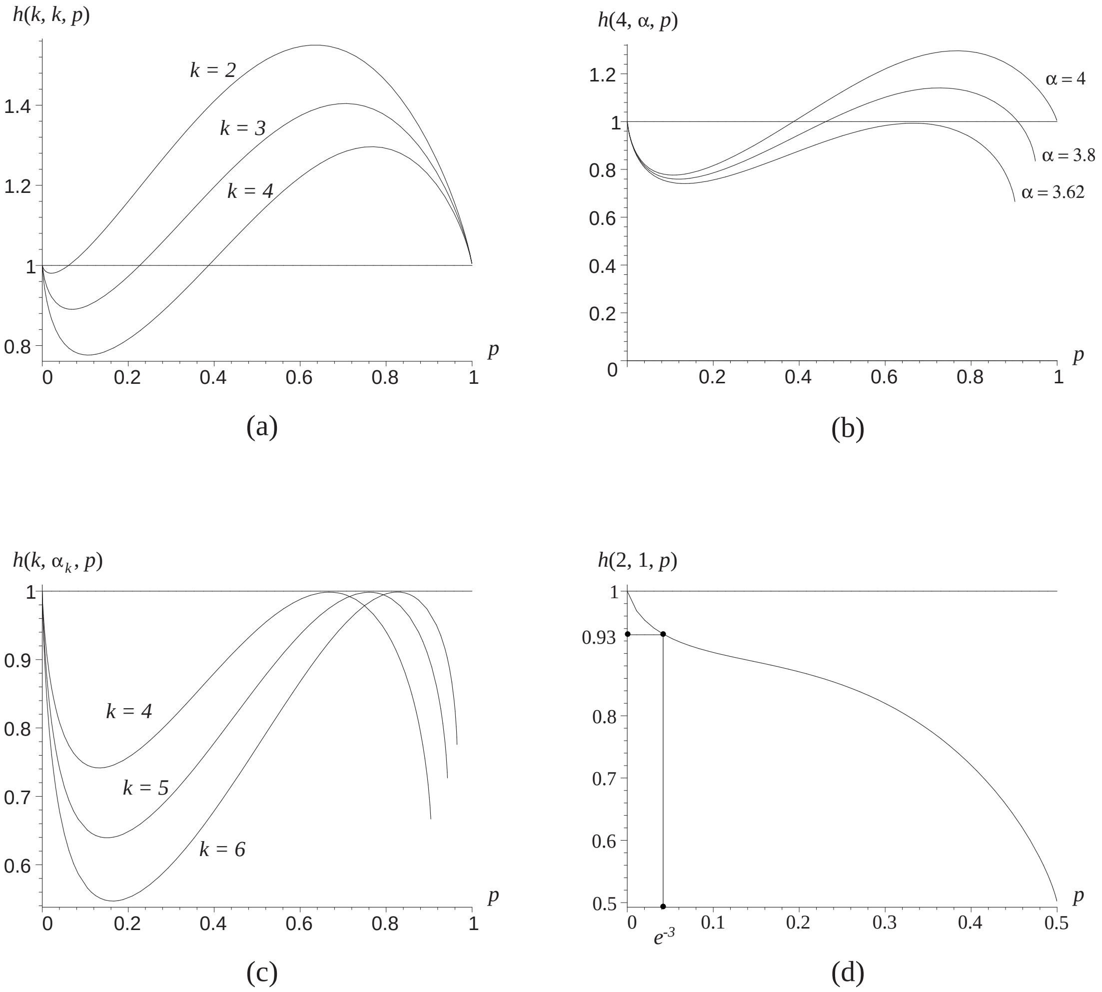

Figure 3.6: Figure (a) shows the functions $h(2,2,p)$, $h_1(3,3,p)$, and $h(4,4,p)$. Figure (b) shows the functions $h(4,4,p)$, $h(4,3.8,p)$, and $h(4,3.62,p)$. Figure (c) shows the functions $h(4,\alpha_4,p)$, $h(5,\alpha_5,p)$, and $h(6,\alpha_6,p)$. Figure (d) shows that the function $h(2,1,p) < 0.93$ on $[e^{-3},1/2)$. The figures illustrate that the function h is strictly decreasing on k, and strictly increasing on α. The function $h(k,\alpha_k,p) \leq 1$, on $(0,\alpha_k/k)$ where the equality holds only at one point. For $\alpha \in (\alpha_k,k]$, the functions $h(k,\alpha,p)$ intersects the line $y = 1$ at two positive points.

Comparing the bounds of Table 3.3 with the ones obtained from the $(k+1)$-core analysis in Table 3.1, we see that Theorem 3.3 is a clear improvement, except for $k = 2, 3$, where $c_2 \geq 1.67545943... > \alpha_2$ and $c_3 \geq 2.57470137... > \alpha_3$. We deal with this problem in Section 3.5.2 where we improve the bounds for $k = 3, 4, 5$, but unfortunately not for $k = 2$. Thus, $c_2 \geq 1.67545943...$ seems to be the best bound so far.

Four Technical Lemmas

Before we start the proof of Theorem 3.3, we need to establish some lemmas. Recall that for $x \in (0, 1)$, we have the following known inequalities:

$$\log(1+x) < x\,, \quad \text{or} \quad \log(x) < x - 1\,, \tag{3.4}$$

and,

$$(1-x)\log(1-x) > -x\,, \quad \text{or} \quad (1-x)^{1-x} > e^{-x}\,. \tag{3.5}$$

The following lemma highlights some of the analytical properties of the function h. Figure 3.6 illustrates some of these properties.

Lemma 3.3. *Let $\tilde{k} > k \geq 2$ be integers, and $\alpha, \tilde{\alpha} \in (0, k]$ be such that $\alpha < \tilde{\alpha}$. The following are true:*

1. *For all $p \in (0, \alpha/k)$, we have $h(k, \alpha, p) < h(k, \tilde{\alpha}, p)$; and if both $\alpha, \tilde{\alpha} > k/e$, then $h(k, \alpha, \alpha/k) < h(k, \tilde{\alpha}, \tilde{\alpha}/k)$.*

2. *For any constant $a \in (0, 1]$, we have $h(k, ak, p) > h(\tilde{k}, a\tilde{k}, p)$, for all $p \in (0, a)$.*

3. *$h(k, \alpha, p) < 1$, where $0 < p \leq \min(e^{-3}, \alpha/k)$.*

4. *There is an $\epsilon \in (0, 1)$, such that $h(k, k, p) > 1$, for all $p \in (1 - \epsilon, 1)$.*

Proof. First, by definition of $f(k,\alpha,p)$, we see that for fixed $p \in (0,\alpha/k)$,

$$\begin{aligned}
\frac{\partial}{\partial\alpha}(\log f) &= \frac{\partial}{\partial\alpha}\big((\alpha-kp)\log(\alpha-\alpha p^2)-(\alpha-kp)\log(\alpha-kp)+kp\log(\alpha p/k)\big)\\
&= \log(\alpha-\alpha p^2)+1-kp/\alpha-\log(\alpha-kp)-1+kp/\alpha\\
&= \log\left(\frac{1-p^2}{1-kp/\alpha}\right)>0\,,
\end{aligned}$$

which is true because $1-p^2>1-kp/\alpha$. Since f is strictly positive, then

$$\frac{\partial}{\partial\alpha}f(k,\alpha,p)=f(k,\alpha,p)\frac{\partial}{\partial\alpha}\log f(k,\alpha,p)>0\,.$$

This means that $f(k,\alpha,p)$, and hence $h(k,\alpha,p)$, is a strictly increasing function of α, where $p\in(0,\alpha/k)$. If $p=\alpha/k$, then

$$h(k,\alpha,\alpha/k)=\left(\frac{\alpha}{k}\right)^{2\alpha}<\left(\frac{\tilde\alpha}{k}\right)^{2\tilde\alpha}=h(k,\tilde\alpha,\tilde\alpha/k)\,,$$

which is true because if $t(x)=2x\log(x/k)$, where $x\in(k/e,k]$, then the derivative $t'(x)=2\log(x/k)+2>0$, i.e., $t(x)$ is a strictly increasing function. Secondly, if $\alpha=ak$, for some constant $a\in(0,1]$, and $p\in(0,a)$, we have

$$f(k,ak,p)=\left(\frac{a(1-p^2)}{a-p}\right)^{k(a-p)}(ap)^{kp}\,.$$

Let $g(a,p)=(a-p)\log(1-p^2)-(a-p)\log(1-p/a)$, and notice that

$$\frac{\partial g}{\partial a}=\log(1-p^2)-\log(1-p/a)+1>0\,,$$

because $1-p^2>1-p/a$. This means that for fixed $p\in(0,a)$, the function $g(a,p)$ strictly increases as a function of a. Thus, using the inequalities in (3.4), we see that

$$\begin{aligned}
\frac{\partial}{\partial k}(\log f(k,ak,p)) &= (a-p)\log\frac{1-p^2}{1-p/a}+p\log(ap)\\
&\le (1-p)\log\frac{1-p^2}{1-p}+p\log p\\
&< p(1-p)+p(p-1)\ \le\ 0\,.
\end{aligned}$$

Therefore, the function $f(k, ak, p)$, and hence $h(k, ak, p)$, strictly decreases on k. Thirdly, we know thus far that for any integer $k \geq 2$, $\alpha \in (0, k]$, and $p \in (0, \alpha/k)$, we have $h(k, \alpha, p) \leq h(k, k, p) \leq h(, 2, 2, p)$. However, using (3.5), we see that for $p \in (0, e^{-3}]$,

$$\begin{aligned} h(2,2,p) &= (1-p)^{-(1-p)}(1+p)^{2(1-p)}p^p \\ &< \exp(p + 2p(1-p) + p\log e^{-3}) \\ &= \exp(-2p^2) \ \leq \ 1\,. \end{aligned}$$

Fourthly, when $\alpha = k$,

$$h(k,k,p) = \frac{(1+p)^{k(1-p)}p^{kp}}{p^p(1-p)^{1-p}}\,,$$

and hence,

$$\begin{aligned} \frac{\partial}{\partial p}(\log h) &= \frac{\partial}{\partial p}\left(k(1-p)\log(1+p) + kp\log p - p\log p - (1-p)\log(1-p)\right) \\ &= -k\log(1+p) + \frac{k(1-p)}{(1+p)} + (k-1)\log p + k - 1 + \log(1-p) + 1\,, \end{aligned}$$

which converges to $-\infty$ as p goes to 1. Since the derivatives of h and $\log h$ have the same sign, this means that $h(k, k, p)$ is strictly decreasing on $(1 - \epsilon, 1)$ for some positive ϵ, i.e., $h(k, k, p) > 1$, for all $p \in (1 - \epsilon, 1)$. $\square$

Since h is continuous on its domain, $h(k, k, e^{-3}) < 1$ (Lemma 3.3), and $h(k, k, q) > 1$ for some $q \in (e^{-3}, 1)$, the equation $h(k, k, p) = 1$ must have a solution in (e^{-3}, q). The following lemma bounds the smallest such solution from below. The lemma helps us later on to establish the lower bound on α_k, and to prove that the smallest k-overloaded set in the random graph $\mathbb{G}(n, kn)$ has size ratio of at least $1 - 2^k \exp\left(-k + 1 + e^{-k/4}\right)$.

Lemma 3.4. *For an integer $k \geq 2$, let r_k be the smallest positive root of the equation $h(k, k, r_k) = 1$. Then for k large enough,*

$$r_k > 1 - \left(\frac{2}{e}\right)^k e^{1+e^{-k/4}}\,.$$

Proof. Let $\rho_k = \exp((\log 2 - 1)k + 1 + e^{-k/4})$. Since h is continuous, and by Lemma 3.3, we have $h(k,k,p) < 1$, for all $p \in (0, e^{-3}]$, then

$$r_k > 1 - \rho_k \iff h(k,k,p) < 1, \text{ for all } p \in (e^{-3}, 1-\rho_k]\,.$$

We shall show that for k large enough, the function $g(p) := \log h(k,k,p) < 0$, for all $p \in (e^{-3}, 1-\rho_k]$. First notice that

$$\begin{aligned} g(p) &= k(1-p)\log(1+p) + (k-1)p\log p - (1-p)\log(1-p)\,, \\ g'(p) &= -k\log(1+p) + \frac{k(1-p)}{1+p} + (k-1)\log p + \log(1-p) + k\,. \end{aligned}$$

Thus,

$$\begin{aligned} & g''(p) = \frac{-k}{1+p} - \frac{2k}{(1+p)^2} + \frac{k-1}{p} - \frac{1}{1-p} = 0 \\ \iff\quad & -kp(1-p^2) - 2kp(1-p) + (k-1)(1+p)^2(1-p) - p(1+p)^2 = 0 \\ \iff\quad & (k-1)p^2 - 2(k+1)p + k - 1 = 0 \\ \iff\quad & p = \frac{k+1-2\sqrt{k}}{k-1} \stackrel{\text{def}}{=} q_k\,. \end{aligned}$$

Evidently, $g''(p)$ is strictly positive on $(0, q_k)$, and negative on $(q_k, 1)$. This yields that $g(p)$ is strictly convex on $(0, q_k)$, and $g'(p)$ is decreasing on $[q_k, 1)$. Moreover, using (3.5), we see that for $p \in [q_k, 1-\rho_k]$,

$$\begin{aligned} g'(p) &\geq g'(1-\rho_k) \\ &> k - k\log(2-\rho_k) + (k-1)\log(1-\rho_k) + \log\rho_k \\ &> (1-\log 2)k - \frac{(k-1)\rho_k}{1-\rho_k} + (\log 2 - 1)k + 1 \\ &= 1 - \frac{k2^k - 2^k}{e^{k-2} - 2^k} > 0\,, \end{aligned}$$

when $k \geq 16$. This means that $g(p)$ is strictly increasing on $[q_k, 1-\rho_k]$. Consequently, $g(p) \leq \max(g(e^{-3}),\, g(1-\rho_k))$, for all $p \in [e^{-3}, 1-\rho_k]$. However, we know that

$g(e^{-3}) < 0$, and for k large enough, $(k \geq 100)$, we have

$$\begin{aligned} g(1-\rho_k) &= k\rho_k \log(2-\rho_k) + (k-1)(1-\rho_k)\log(1-\rho_k) - \rho_k \log \rho_k \\ &< k\rho_k \log 2 - (k-1)\rho_k(1-\rho_k) - (\log 2 - 1)k\rho_k - \rho_k(1+e^{-k/4}) \\ &= \rho_k\left((k-1)\rho_k - e^{-k/4}\right) < 0\,, \end{aligned}$$

which completes the proof. □

Next, we turn our attention to the definition of α_k. Let

$$\mathcal{A} := \left\{\alpha > 0 \,:\, \exists\, \delta \in (0,1) \text{ such that } h(k,\alpha,p) \leq \delta, \;\; \forall\, p \in (e^{-3}, \alpha/k)\right\}\,,$$

and recall that $\alpha_k = \sup \mathcal{A}$. Clearly, if $\beta \in \mathcal{A}$, then $(0,\beta) \subseteq \mathcal{A}$, because h is an increasing function of α. Also, if $\gamma \notin \mathcal{A}$, then $\alpha_k \leq \gamma$. This leads to $\alpha_k \leq k$, because $h(k,k,1) = 1$. The following lemma follows easily.

Lemma 3.5. *For any fixed integer $k \geq 2$, α_k is well-defined and $\alpha_k \in (k/2,\, k)$. Moreover, $h(k,\alpha_k,p) \leq 1$, for all $p \in [0,\alpha_k/k]$.*

Proof. First, α_k is well-defined because $\mathcal{A} \neq \varnothing$. For instance, by Lemma 3.3, we have $h(k,k/2,p) < h(2,1,p) < 0.93$, for $p \in (e^{-3},1/2)$ (see Figure 3.6-(d)), and hence $k/2 \in \mathcal{A}$. Thus, trivially, $\alpha_k \geq k/2$. Now notice that $h(k,\alpha_k,0) = 1$, and $h(k,\alpha_k,\alpha_k/k) \leq 1$. So if possible, assume that there is a point $q \in (0,\alpha_k/k)$ such that $h(k,\alpha_k,q) > 1$. By the definition of h, we have $h(k,qk,q) = q^{2qk} < 1$. Therefore, since h is a continuous increasing function of α, then there is $\tilde{\alpha} \in (qk,\alpha_k)$ such that $h(k,\tilde{\alpha},q) = 1$. That is, $\tilde{\alpha} \notin \mathcal{A}$, and hence $\alpha_k \leq \tilde{\alpha}$ which is a contradiction. Thus, $h(k,\alpha_k,p) \leq 1$, for all $p \in (0,\alpha_k/k)$. Consequently, Lemma 3.3-(4) leads to $\alpha_k < k$. □

Finally, we have the following lemma.

Lemma 3.6. *Let $k \geq 2$ be any fixed integer. The following are true:*

1. *For $\alpha \in (\alpha_k, k]$, the equation $h(k, \alpha, p) = 1$ has at least two positive solutions, and there exists a point $s(k, \alpha) \in (e^{-3}, \alpha/k)$ such that*

$$\max_{0 \le p \le \alpha/k} h(k, \alpha, p) = h(k, \alpha, s) > 1\,.$$

2. *The equation $h(k, \alpha_k, p) = 1$ has at least one positive solution.*

3. *For $\alpha, \tilde{\alpha} \in [\alpha_k, k]$, if $r(k, \alpha)$ is the smallest positive solution of $h(k, \alpha, p) = 1$, then $r(k, \tilde{\alpha}) > r(k, \alpha)$, whenever $\alpha > \tilde{\alpha}$.*

Proof. First, for $\alpha \in (\alpha_k, k]$, let $s(k, \alpha)$ be any point at which $h(k, \alpha, p)$ attains its maximum on $[e^{-3},\ \alpha/k]$, and let $\lambda = h(k, \alpha, s)$, which is positive. If possible, assume that $\lambda \le 1$. Let $\beta = (\alpha + \alpha_k)/2$. Notice that $k/2 \le \alpha_k < \beta < \alpha$. Let q be any point at which $h(k, \beta, p)$ attains its maximum on $[e^{-3}, \beta/k]$. Then by Lemma 3.3, we see that for all $p \in [e^{-3}, \beta/k]$,

$$h(k, \beta, p) \le \delta \stackrel{\text{def}}{=} h(k, \beta, q) < h(k, \alpha, q) \le \lambda \le 1\,.$$

Thus, the definition of α_k yields that $\beta \le \alpha_k$ which is a contradiction. Consequently, $\lambda > 1$. Since $h(k, \alpha, \alpha/k) \le 1$, and by lemma 3.3, $h(k, \alpha, p) \le 1$, for all $p \in [0, e^{-3}]$, then $s \in (e^{-3}, \alpha/k)$, and

$$\max_{0 \le p \le \alpha/k} h(k, \alpha, p) = h(k, \alpha, s) > 1\,.$$

Since $h(k, \alpha, s) > 1$, and $h(k, \alpha, e^{-3}) < 1$, then there is a point $q_1(k, \alpha) \in (e^{-3}, s)$ such that $h(k, \alpha, q_1) = 1$, because h is continuous. If $\alpha = k$, we know that $h(k, \alpha, \alpha/k) = 1$; and if $\alpha < k$, we have $h(k, \alpha, \alpha/k) < 1$, and hence—for the same reason again—there is a point $q_2(k, \alpha) \in (s, \alpha/k)$ such that $h(k, \alpha, q_2) = 1$. That is, $h(k, \alpha, p) = 1$ has at least two positive solutions. Next, let

$$\sigma_k := \lim_{\alpha \searrow \alpha_k} s(k, \alpha)\,.$$

and notice that

$$h(k,\alpha_k,\sigma_k) = \lim_{\alpha\searrow\alpha_k} h(k,\alpha,s) \geq 1\,,$$

because h is continuous on each of its arguments. However, by Lemma 3.5, we have $h(k,\alpha_k,p) \leq 1$, for all $p \in [0,\alpha_k/k]$. Thus, $h(k,\alpha_k,\sigma_k) = 1$. Finally, Lemma 3.3-(3) yields that $h(k,\alpha,p) < h(k,\alpha,r(k,\alpha)) = 1$, for $p \in (0,r(k,\alpha))$. Since $h(k,\alpha,p)$ is an increasing function of α, then for any $\tilde{\alpha} \in [\alpha_k,\alpha)$, we have $h(k,\tilde{\alpha},p) < h(k,\alpha,p) \leq 1$, for all $p \in (0,r(k,\alpha)]$. This means that $r(k,\tilde{\alpha}) > r(k,\alpha)$. □

Proof of Theorem 3.3.

First, we prove that $c_k \geq \alpha_k$. Let $\epsilon \in (0,1)$ be any small arbitrary constant. Let $\beta = \alpha_k - \epsilon$. By Lemma 3.2 and since ϵ is arbitrary, it suffices to show that the random graph $\mathbb{G}(n,\lfloor \beta n \rfloor)$ does not contain any k-overloaded set of size $\geq e^{-3}n$. For $\lfloor e^{-3}n \rfloor \leq i \leq \lfloor (\beta n-1)/k \rfloor$, let $p_i := i/n$, and notice that $p_i \in (e^{-3},\beta/k)$. By the definition of α_k, there exists $\alpha > \beta$, and a constant $\delta \in (0,1)$ such that $h(k,\alpha,p) \leq \delta$, for all $p \in (e^{-3},\alpha/k)$. Since h is an increasing function of α, then $h(k,\beta,p) \leq h(k,\alpha,p) \leq \delta$, for all $p \in (e^{-3},\beta/k)$. Thus, using inequality (1) of Lemma 0.1, we see that the probability that $\mathbb{G}(n,\lfloor \beta n \rfloor)$ contains a k-overloaded set of size at least $e^{-3}n$ is not more than

$$\begin{aligned}
\sum_{i=\lfloor e^{-3}n \rfloor}^{\lfloor (\beta n-1)/k \rfloor} \binom{n}{i} \mathbb{P}\left\{\mathrm{Bin}(\lfloor \beta n \rfloor, i^2/n^2) > ki\right\} &\leq \sum_{i=\lfloor e^{-3}n \rfloor}^{\lfloor (\beta n-1)/k \rfloor} \frac{n^n}{i^i(n-i)^{(n-i)}} \Upsilon(kp_i/\beta, p_i^2)^{\beta n} \\
&= \sum_{i=\lfloor e^{-3}n \rfloor}^{\lfloor (\beta n-1)/k \rfloor} h(k,\beta,p_i)^n \\
&\leq n\,\delta^n = o(1)\,.
\end{aligned}$$

Secondly, by Lemma 3.5, $h(k,\alpha_k,p) \leq 1$, for all $p \in [0,\alpha_k/k]$. Recall that s_k is a point at which $h(k,\alpha_k,p)$ attains its maximum on $[e^{-3},\alpha_k/k]$. From Lemma

3.6 we know that $h(k, \alpha_k, p) = 1$ has a solution, and thus, $h(k, \alpha_k, s_k) = 1$. Since $\alpha_k < k$, then by definition, $h(k, \alpha_k, \alpha_k/k) < 1$. That is, $s_k \in (0, \alpha_k/k)$ which leads to $h_p(k, \alpha_k, s_k) = 0$, because h is smooth on the open interval.

Thirdly, we know, by Lemma 3.6, that for $\alpha \in (\alpha_k, k]$, the equation $h(k, \alpha, p) = 1$ has at least two positive solutions. Notice that if $c_k = \alpha_k$, there is at least one solution for $h(k, c_k, p) = 1$, namely s_k, and we may have $q_1(k, c_k) = s_k = q_2(k, c_k)$. Nevertheless, the following is still true. Since $h(k, c_k, e^{-3}) < 1$, and $h(k, c_k, c_k/k) < 1$, then the definition of the two points $q_1(k, c_k)$ and $q_2(k, c_k)$ implies that $h(k, c_k, p) < 1$, for all $p \in (0, q_1) \cup (q_2, c_k/k)$. This means that for any arbitrary constant $\epsilon \in (0, 1)$ sufficiently small, there exists a constant $\delta \in (0, 1)$ such that $h(k, c_k, p) < \delta$ for all $p \in (e^{-3}, q_1 - \epsilon] \cup [q_2 + \epsilon, \alpha/k)$. Therefore, using similar argument as above, we conclude that the random graph $\mathbb{G}(n, \lceil c_k n \rceil)$ does not contain any k-overloaded set of size less than $q_1(k, c_k)$ nor greater than $q_2(k, c_k)$, w.h.p. Finally, Lemmas 3.6-(3) and 3.4 lead to

$$\alpha_k/k > s_k \geq q_1(k, \alpha_k) \geq q_1(k, c_k) > q_1(k, k) = r_k > 1 - e^{1+e^{-k/4}} (2/e)^k \,,$$

for k large enough. $\square$

3.5.2 Further Improvements

The lower bounds α_k, for $k = 2, 3$, are smaller than the corresponding ones of the $(k + 1)$-core thresholds. In the following, we improve the lower bounds on c_k, for $k = 3, 4, 5$. However, the technique we use here is not helpful enough to beat the 1.67545943... threshold of the 3-core which stays the best lower bound, thus far, on c_2. We utilize the following lemma to tighten the analysis. Recall that the degree of any vertex is defined to be the number of its non-loop incident edges plus twice the number of its loops. For a set of vertices S we write, throughout, $\min\deg(S)$ to denote the minimum degree—restricted to the subgraph $(S, \mathcal{E}(S))$—of any vertex in S.

Lemma 3.7. *Let $k \in \mathbb{N}$. In any graph G, if $S \subseteq \mathcal{V}(G)$ is a k-overloaded set such that $|\mathcal{E}(A)| \leq k\,|A|$, for any proper subset $A \subset S$, then* $\min\deg(S) \geq k+1$.

Proof. Let v be any vertex in S, and define $A = S - \{v\}$. Since $|\mathcal{E}(A)| \leq k\,|A|$, and $|\mathcal{E}(S)| \geq k\,|S| + 1$, then

$$\deg(v) \geq |\mathcal{E}(S)| - |\mathcal{E}(A)| \geq k(|A|+1) + 1 - k\,|A| = k+1\,.$$

□

The lemma says that the minimum degree of the smallest k-overloaded set is at least $k+1$, because it does not contain any k-overloaded proper subset. That is, the smallest k-overloaded set is a connected subgraph where every vertex has degree of at least $k+1$. Otherwise, it consists of two disjoint sets that are not k-overloaded, and hence their union is also not a k-overloaded set. Thus, if the graph has a k-overloaded set then it has a $(k+1)$-core, and its size is at least the size of the smallest k-overloaded set in the graph. Now we are ready to prove the following improved lower bounds on c_k.

k	β_k	α_k
3	2.61845509...	2.48312473...
4	3.65354252...	3.61901095...
5	4.71959504...	4.71902985...

Table 3.5: The threshold $c_k \geq \beta_k > \alpha_k$.

Theorem 3.4. *For $k = 3, 4, 5$, let β_k be as specified in Table 3.5. Then the threshold $c_k \geq \beta_k$.*

Proof. First, we show how Lemma 3.7 can be used to tighten the analysis. Suppose we want to prove that the random graph $\mathbb{G}(n,m)$ is k-orientable, for given $m < kn$. By Lemma 3.2, it suffices to show that $\mathbb{G}(n,m)$ does not contain any k-overloaded set of size greater than $i_0 := \lceil ne^{-3} \rceil$. Let A be the event that the random graph $\mathbb{G}(n,m)$ is not k-orientable. For $i = 1, \ldots, n$, let C_i be the event that $\mathbb{G}(n,m)$ does not contain any k-overloaded set of size $\le i$, and B_i be the event that it contains a k-overloaded set of size i. Notice that $\mathbb{P}\left\{C_{i_0-1}^c\right\} = o(1)$ (Lemma 3.2), and that

$$\mathbb{P}\{A\} \le \mathbb{P}\{A \cap C_{i_0-1}\} + \mathbb{P}\left\{C_{i_0-1}^c\right\} .$$

Suppose that the vertices in $\mathbb{G}(n,m)$ are numbered $1, \ldots, n$. For $1 \le j \le i \le n$, let $S_i := \{1, \ldots, i\}$, and let $D_i(j) = [\, d_i(j) > k \,]$, where $d_i(j)$ is the degree of the j-th vertex restricted to the subgraph $(S_i, \mathcal{E}(S_i))$. Suppose that $m = \lfloor \beta n \rfloor$, where $\beta \in (\alpha_k, k)$ is a constant to be chosen later on to be as large as possible. Let $i^* = \lfloor \beta n/k \rfloor$. Using Lemmas 3.1 and 3.7, we see that for n large enough,

$$\begin{aligned}
\mathbb{P}\{A \cap C_{i_0-1}\} &= \mathbb{P}\left\{\left(\bigcup_{i=i_0}^{i^*} B_i\right) \cap C_{i_0-1}\right\} \\
&= \mathbb{P}\{B_{i_0} \cap C_{i_0-1}\} + \sum_{i=i_0+1}^{i^*} \mathbb{P}\left\{B_i \cap B_{i-1}^c \cap \ldots \cap B_{i_0}^c \cap C_{i_0-1}\right\} \\
&= \sum_{i=i_0}^{i^*} \mathbb{P}\{B_i \cap C_{i-1}\} \;\le\; \sum_{i=i_0}^{i^*} \binom{n}{i} \mathbb{P}\{\,[\,|\mathcal{E}(S_i)| > ki\,] \cap C_{i-1}\} \\
&\le \sum_{i=i_0}^{i^*} \binom{n}{i} \mathbb{P}\{\,[\,|\mathcal{E}(S_i)| > ki\,] \cap [\,\min\deg(S_i) > k\,]\,\} \\
&\le \sum_{i=i_0}^{\lfloor bn \rfloor} \binom{n}{i} (x_i + y_i) \;+\; \sum_{i=\lceil bn \rceil}^{i^*} \binom{n}{i} \mathbb{P}\{|\mathcal{E}(S_i)| > ki\} ,
\end{aligned}$$

where $b \in (e^{-3}, \beta/k)$ is a constant to be picked later, and x_i and y_i are the following probabilities:

$$\begin{aligned}
x_i &= \mathbb{P}\{[\,\min\deg(S_i) > k\,]\} \cap [\,ki < |\mathcal{E}(S_i)| \le (bn+i)k/2\,] \\
y_i &= \mathbb{P}\{|\mathcal{E}(S_i)| > (bn+i)k/2\} .
\end{aligned}$$

We estimate the probability x_i as follows. Recall that $d_i(j)$, the degree of the j-th vertex in the subgraph $(S_i, \mathcal{E}(S_i))$, is a binomial random variable with parameters $2\,|\mathcal{E}(S_i)|$ and $1/n$. Using a conditional probability argument, we see that

$$\begin{aligned}
\mathbb{P}\left\{\cap_{j=1}^{i} D_i(j) \;\middle|\; |\mathcal{E}(S_i)| = r\right\} &= \mathbb{P}\left\{D_i(1) \mid |\mathcal{E}(S_i)| = r\right\} \times \\
&\quad \mathbb{P}\left\{D_i(2) \mid [\,|\mathcal{E}(S_i)| = r\,] \cap D_i(1)\right\} \times \cdots \times \\
&\quad \mathbb{P}\left\{D_i(i) \mid [\,|\mathcal{E}(S_i)| = r\,] \cap D_i(1) \cap \ldots \cap D_i(i-1)\right\} \\
&= \prod_{j=0}^{i-1} \mathbb{P}\left\{\mathrm{Bin}(2r - j(k+1), 1/n) > k\right\} .
\end{aligned}$$

Since $|\mathcal{E}(S_i)| \stackrel{\mathcal{L}}{=} \mathrm{Bin}(m, i^2/n^2)$, we have

$$\begin{aligned}
x_i &= \sum_{r=ki+1}^{\lfloor (bn+i)k/2 \rfloor} \mathbb{P}\left\{|\mathcal{E}(S_i)| = r\right\} \mathbb{P}\left\{\cap_{j=1}^{i} D_i(j) \;\middle|\; |\mathcal{E}(S_i)| = r\right\} \\
&\le \sum_{r=ki+1}^{\lfloor (bn+i)k/2 \rfloor} \mathbb{P}\left\{\mathrm{Bin}(m, i^2/n^2) = r\right\} \prod_{j=0}^{i-1} \mathbb{P}\left\{\mathrm{Bin}(2r - j(k+1), 1/n) > k\right\} \\
&\le \sum_{r=ki+1}^{\lfloor (bn+i)k/2 \rfloor} \mathbb{P}\left\{\mathrm{Bin}(m, i^2/n^2) = r\right\} \prod_{j=j_0}^{i-1} \mathbb{P}\left\{\mathrm{Bin}(2r - j(k+1), 1/n) > k\right\} ,
\end{aligned}$$

where $j_0 = \max\left(\lceil (2r - bnk)/(k+1) - \gamma_n \rceil, 0\right)$, and $\gamma_n = o(n) \xrightarrow{n} \infty$. Notice that

$$0 \le j_0 \le \frac{k(i+bn)}{k+1} - \frac{bnk}{k+1} - \gamma_n + 1 \le \frac{ki}{k+1} ,$$

and for all $j \ge j_0$, we have $(2r - j(k+1))/n < bk < k$. Now if $i \in [i_0, bn/2]$, and $r \in (ki, bnk/2]$, we have $j_0 = 0$; and thence, Angluin-Valiant's inequality implies

$$\begin{aligned}
\prod_{j=j_0}^{i-1} \mathbb{P}\left\{\mathrm{Bin}(2r - j(k+1), 1/n) > k\right\} &\le \mathbb{P}\left\{\mathrm{Bin}\left(\lfloor bnk \rfloor, 1/n\right) > k\right\}^{i-1} \\
&\le \exp\left(-bk(i-1)\left(1 - \frac{1}{b} + \frac{1}{b}\log\frac{1}{b}\right)\right) \\
&\le \exp\left(-k(i-1)(b - \log b - 1)\right) \\
&\le \exp\left(-\frac{ki(b - \log b - 1)}{k+1}\right) ,
\end{aligned}$$

because the constant $b - \log b - 1$ is always positive, by inequality (3.4). On the other hand, if $i \in [i_0,\, bn/2]$, and $r \in (bnk/2,\, (bn+i)k/2]$, or $i \in (bn/2,\, bn)$, we have $j_0 = \lceil\, (2r - bnk)/(k+1) - \gamma_n \,\rceil$. Using the Angluin-Valiant inequality again, we see that

$$
\begin{aligned}
\prod_{j=j_0}^{i-1} \mathbb{P}\left\{\mathrm{Bin}(2r - j(k+1), 1/n) > k\right\} &\le \mathbb{P}\left\{\mathrm{Bin}\left(\lfloor\, bnk + (k+1)\gamma_n \,\rfloor, 1/n\right) > k\right\}^{i-j_0} \\
&\le \exp\left(-k(i - j_0)(b - \log b - 1 + o(1))\right) \\
&\le \exp\left(-\frac{ki(b - \log b - 1)}{k+1}\right).
\end{aligned}
$$

Now set $p_i = i/n$, and observe that by inequality (1) of Lemma 0.1, we see that for all $r \in (ki, (bn+i)k/2]$,

$$
\begin{aligned}
\mathbb{P}\left\{\mathrm{Bin}(m, i^2/n^2) = r\right\} &\le \mathbb{P}\left\{\mathrm{Bin}(m, i^2/n^2) \ge r\right\} \\
&\le \mathbb{P}\left\{\mathrm{Bin}(m, i^2/n^2) \ge ki\right\} \\
&\le \Upsilon(kp_i/\beta,\, p_i^2)^{\beta n} = f(k, \beta, p_i)^n\,,
\end{aligned}
$$

where the function f is as defined in (3.2). For convenience, let

$$
g(k, \beta, p) := f(k, \beta, p)\, \exp\left(\frac{-kp(b - \log b - 1)}{k+1}\right),
$$

and notice that g strictly increases as β does, because f does. Thus far, we have

$$
\begin{aligned}
\sum_{i=i_0}^{\lfloor\, bn\,\rfloor} \binom{n}{i} x_i &\le \sum_{i=i_0}^{\lfloor\, bn/2\,\rfloor} \binom{n}{i} m\, g(k, \beta, p_i)^n + \sum_{i=\lceil\, bn/2\,\rceil}^{\lfloor\, bn\,\rfloor} \binom{n}{i} m\, g(k, \beta, p_i)^n \\
&\le m\, 2^{bn/2} \max_{e^{-3} \le p \le b/2} g(k, \beta, p)^n + m\, 2^{bn} \max_{b/2 \le p \le b} g(k, \beta, p)^n\,. \qquad (3.6)
\end{aligned}
$$

Next, we use inequality (1) to bound the probability y_i, so that

$$
\begin{aligned}
\sum_{i=i_0}^{\lfloor\, bn\,\rfloor} \binom{n}{i} y_i &= \sum_{i=i_0}^{\lfloor\, bn\,\rfloor} \binom{n}{i} \mathbb{P}\left\{\mathrm{Bin}(m, i^2/n^2) > (bn+i)k/2\right\} \\
&\le \sum_{i=i_0}^{\lfloor\, bn\,\rfloor} \binom{n}{i} t(k, \beta, p_i)^n \\
&\le 2^{bn} \max_{e^{-3} \le p \le b} t(k, \beta, p)^n\,, \qquad (3.7)
\end{aligned}
$$

where

$$
\begin{aligned}
t(k,\beta,p) &:= \Upsilon\left(\frac{(b+p)k}{2\beta},\, p^2\right)^{\beta} \\
&= \left(\frac{2\beta(1-p^2)}{2\beta-(b+p)k}\right)^{\beta-(b+p)k/2}\left(\frac{2\beta p^2}{(b+p)k}\right)^{(b+p)k/2},
\end{aligned}
$$

and $p \in [e^{-3}, b]$. Notice that $t(k,\beta,p)$ is a strictly increasing function of β for the similar reason mentioned in the proof of the first part of Lemma 3.3. Finally, we use the function h defined in (3.3), to bound the last part just like before,

$$
\sum_{i=\lceil bn \rceil}^{i^*} \binom{n}{i} \mathbb{P}\left\{|\mathcal{E}(S_i)| > ki\right\} \le \sum_{i=\lceil bn \rceil}^{i^*} h(k,\beta,p_i)^n \le n \max_{b \le p \le \beta/k} h(k,\beta,p)^n . \tag{3.8}
$$

Now to make the inequalities (3.6), (3.7) and (3.8) converge to zero, we choose the constant b such that the following conditions are satisfied with the largest possible $\beta \in (\alpha_k, k)$:

$$
\begin{aligned}
&\max_{e^{-3} \le p \le b/2} g(k,\beta,p) < 2^{-b/2}, && \max_{b/2 \le p \le b} g(k,\beta,p) < 2^{-b}, \\
&\max_{e^{-3} \le p \le b} t(k,\beta,p) < 2^{-b}, \quad \text{and} && \max_{b \le p \le \beta/k} h(k,\beta,p) < 1 .
\end{aligned} \tag{3.9}
$$

Since for any $\beta < k$, we have $h(k,\beta,\beta/k) < 1$, then clearly, in order to satisfy the fourth condition, b must be greater than the largest zero of $h(k,\beta,p) = 1$. Analogously, since t is continuous on its domain, the third condition is satisfied only if b is chosen such that it is strictly less than the smallest zero of $t(k,\beta,p) = 2^{-b}$ on $(e^{-3}, \beta/k)$, and $t(k,\beta,e^{-3}) < 2^{-b}$. Thus, we do the following for all $k = 3, 4, 5$. First, we solve (numerically) the two equations $h(k,\beta,b) = 1$, and $t(k,\beta,b) = 2^{-b}$, simultaneously, where we obtain β_k—as specified in Table 3.5—and b. Notice that $t(k,\beta,b) = f(k,\beta,b)$, and

$$
h(k,\beta,b) = \frac{f(k,\beta,b)}{b^b(1-b)^{1-b}};
$$

and therefore, $b^b(1-b)^{1-b} = 2^{-b}$, which means that $b = 0.772907804\ldots.$ Observe that $b \in (e^{-3}, \beta_k/k)$. Next, we need to prove that β_k and b satisfy the first two conditions, and for any arbitrary $\epsilon \in (0,1)$, $\beta_k - \epsilon$ and b satisfy the last two conditions. This will imply that the random graph $\mathbb{G}(n, \lfloor (\beta_k - \epsilon)n \rfloor)$ is k-orientable, for any arbitrary $\epsilon \in (0,1)$, and hence, $c_k \geq \beta_k$. The first two conditions can be verified by using the classical calculus tools like the derivative tests and the Taylor series expansions. However, instead of burdening the reader with tedious computations, we refer him to Figures 3.7, 3.8 and 3.9 where these functions are drawn. For those who do not mind reading such technical details, they may see the appendix at the end of the thesis. It is not difficult also to prove that $t(k, \beta_k, p)$ is an increasing function in p on $[e^{-3}, b]$, and $h(k, \beta_k, p)$ is a decreasing function in p on $[b, \beta_k/k]$. This implies $h(k, \beta_k, p) \leq 1$ on $[b, \beta_k/k]$, and $t(k, \beta_k, p) \leq 2^{-b}$ on $[e^{-3}, b]$. If q is a point at which $t(k, \beta_k - \epsilon, p)$ attains its maximum on $[e^{-3}, b]$, then

$$\max_{e^{-3} \leq p \leq b} t(k, \beta_k - \epsilon, p) = t(k, \beta_k - \epsilon, q) < t(k, \beta_k, q) \leq 2^{-b}\,.$$

Similarly,

$$\max_{b \leq p \leq \beta/k} h(k, \beta_k - \epsilon, p) < 1\,.$$

□

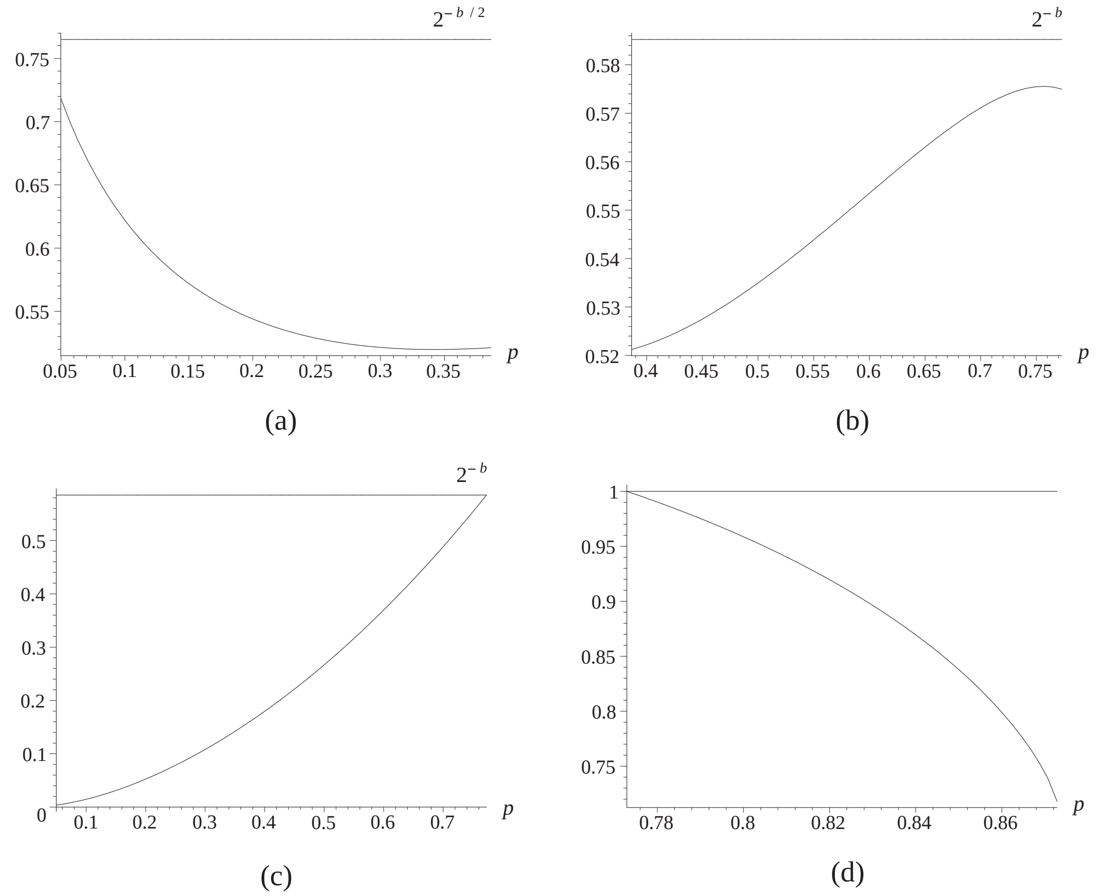

Figure 3.7: Figure (a) shows $g(3, \beta_3, p) < 2^{-b/2}$ on $[e^{-3}, b/2]$. Figure (b) shows $g(3, \beta_3, p) < 2^{-b}$ on $[b/2, b]$. Figure (c) shows $t(3, \beta_3, p) \leq 2^{-b}$ on $[e^{-3}, b]$. Figure (d) shows $h(3, \beta_3, p) \leq 1$ on $[b, \beta_3/3]$.

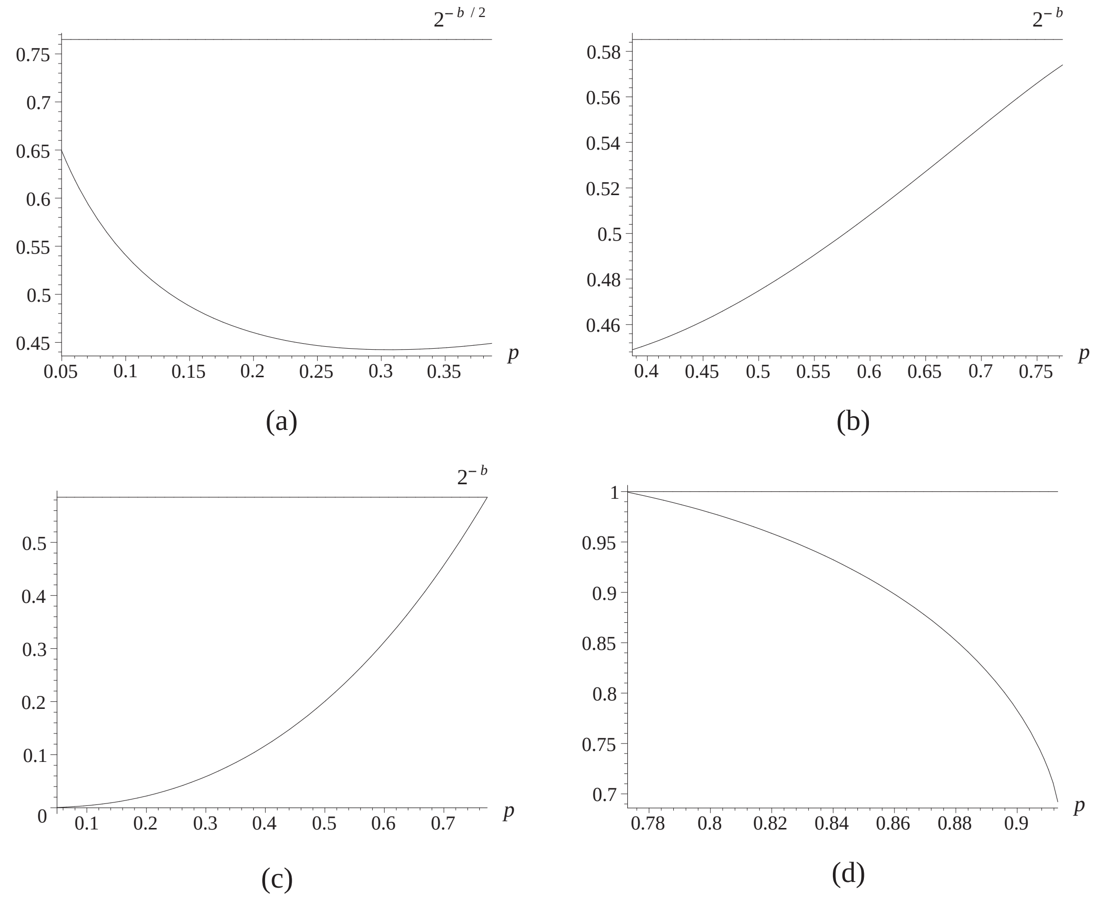

Figure 3.8: Figure (a) shows $g(4, \beta_4, p) < 2^{-b/2}$ on $[e^{-3}, b/2]$. Figure (b) shows $g(4, \beta_4, p) < 2^{-b}$ on $[b/2, b]$. Figure (c) shows $t(4, \beta_4, p) \leq 2^{-b}$ on $[e^{-3}, b]$. Figure (d) shows $h(4, \beta_4, p) \leq 1$ on $[b, \beta_4/4]$.

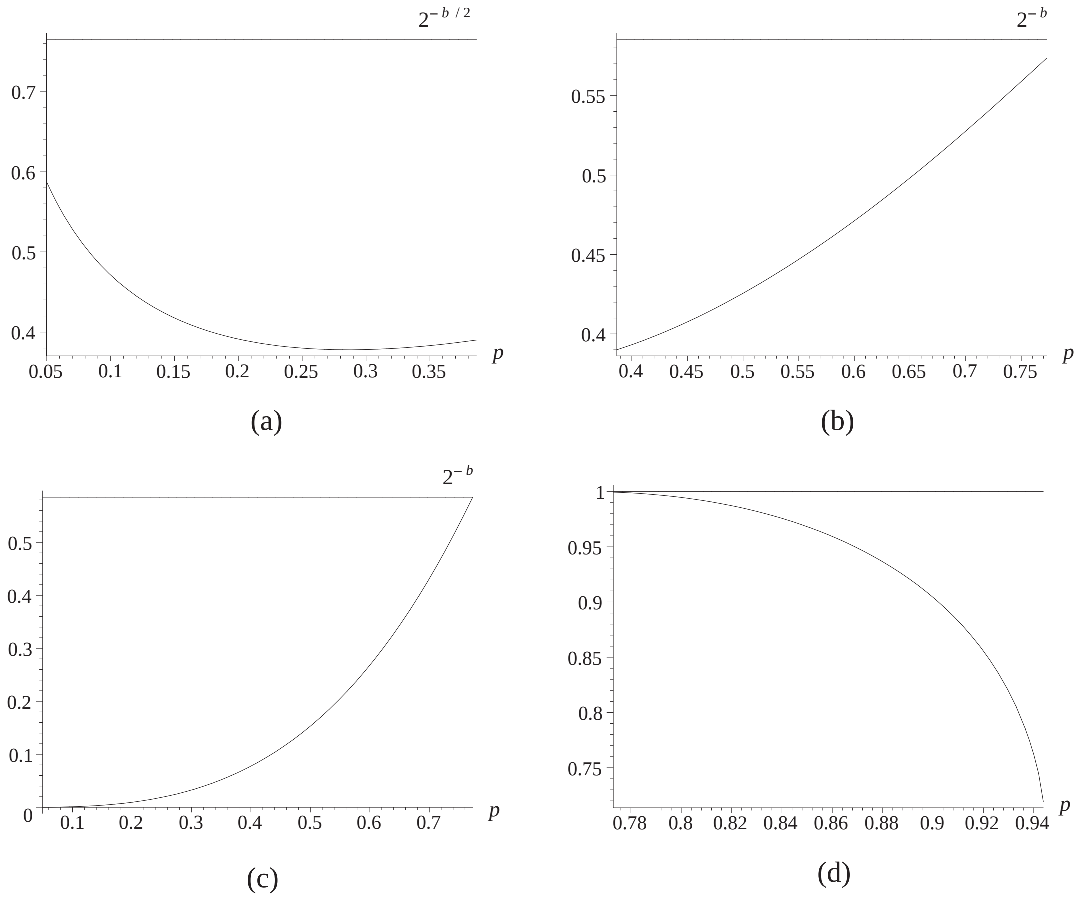

Figure 3.9: Figure (a) shows $g(5, \beta_5, p) < 2^{-b/2}$ on $[e^{-3}, b/2]$. Figure (b) shows $g(5, \beta_5, p) < 2^{-b}$ on $[b/2, b]$. Figure (c) shows $t(5, \beta_5, p) \leq 2^{-b}$ on $[e^{-3}, b]$. Figure (d) shows $h(5, \beta_5, p) \leq 1$ on $[b, \beta_5/5]$.

Chapter 4

Speedups and Trade-offs

The performance of the multiple-choice allocation process can be tuned up in many different ways. In this chapter, we suggest some speedups of algorithm Uniform-GreedyMC, and we study the trade-offs in terms of maximum bin load, insertion (allocation) time, and memory size. For convenience, we present some of our results in terms of balls and bins. Obviously, all of the results, which are summarized in Table 4.1, can be viewed in the context of hashing with chaining in general, and can be applied to algorithm Uniform-ShortChain in particular.

One of the main factors that affects the performance of some of the heuristics we study in this chapter is the number of balls above a certain level, where we say that a ball is at level i if it is inserted into a bin that contains exactly $i-1$ balls just before insertion. Obviously, if $m = n$, then upon termination of algorithm Uniform-GreedyMC(n, m), the number of bins of load at least $L_n = o(\log \log n)$ is at most n/L_n. Thus, the probability that any ball chooses two bins whose loads are at least L_n is at most $1/L_n^2$. Using a binomial tail inequality, we see that the number of balls above the level L_n is $O(n/L_n^2)$, w.h.p. However, we can do better. Recalling the waiting time argument (Theorems 1.3 and 2.2), we notice that at the k-th stage we wait until there are at least n_k bins of load at least k, where

$$n_k = \begin{cases} \lfloor an/2 \rfloor \,, & \text{if } k = 0\,; \\ \left\lfloor an2^{k+\kappa-1} \Big/ 2^{2^{k+\kappa}} \right\rfloor, & \text{for } k \geq 1\,, \end{cases}$$

for some constant $a \in (0,1)$, and an integer $\kappa > 1$ which is a function of $\alpha := m/n$. This means that upon termination of algorithm Uniform-GreedyMC(n,m), and for k large enough, there are more than $n \cdot 2^{-2^{k+\kappa}} = n\exp(-2^{k+\kappa+\log_2 \log 2})$ bins of load at least k, w.h.p. Along the same line, Berenbrink et al. [18] showed this stronger result which, indeed, implies Theorem 0.3. Recall that we write Uniform-GreedyMC(n,m,d), where $d \geq 2$ is an integer, which could be a function of n, to denote the algorithm that inserts m balls into n bins where each ball is placed into the least full bin among d bins chosen independently and uniformly at random, with replacement, breaking ties randomly. If $d = 2$, we often just write Uniform-GreedyMC(n,m).

Theorem 4.1 (Berenbrink et al. [18]). *Let $n, m, k, d \in \mathbb{N}$ such that $m \geq n$, and $d \geq 2$. Let X_k denote the number of bins of load at least $\lfloor k + m/n \rfloor$ upon termination of algorithm* Uniform-GreedyMC(n,m,d). *Then there are some positive constants b and c such that* $\exp(-d^{k+b}) \leq X_k/n \leq \exp(-d^{k-c})$*, w.h.p., for all $k > c$. Furthermore,*

$$\mathbb{P}\left\{X_k > n\exp(-d^{k-c})\right\} = O(1/n^2)\,.$$

Now we have the following corollary.

Corollary 4.1. *For $n, m, k, d \in \mathbb{N}$, where $m \geq n$ and $d \geq 2$, let Y_k be the number of balls above the level $\lfloor k + m/n \rfloor$ upon termination of algorithm* Uniform-GreedyMC(n,m,d). *Then* $\exp(-d^{k+b+1}) \leq Y_k/n \leq \exp(-d^{k-c})$*, w.h.p., for all $k > c$, where b and c are the same constants in Theorem 4.1. Moreover,*

$$\mathbb{P}\left\{Y_k > n\exp(-d^{k-c})\right\} = o(1/n)\,.$$

Proof. For $i \in [m]$, let X_i be the number of bins of load at least $\lfloor i + m/n \rfloor$. First, Theorem 4.1 implies that $Y_k \geq X_{k+1} \geq n\exp(-d^{k+b+1})$, w.h.p. Now for the upper bound, let $A_i = [\, X_i \leq n\exp(-d^{i-c}) \,]$. Let $K = \lceil \log_d \log n + c + 1 \rceil$. Notice that if A_K is true, then for all $i \geq K$, the event A_i is also true. Let $\tilde{k} := \exp(d^{k-c})$, and observe that $\tilde{k} \geq 2$, because $k > c$. Thus, if all the events A_i, for $i = k+1, \ldots, K$, are true, we see that

$$Y_k = \sum_{i=k+1}^{m} X_i \leq \sum_{i=1}^{m} n\exp\left(-d^{k-c+i}\right) = \frac{n}{\tilde{k}} \sum_{i=1}^{m} \exp\left(-(d^i - 1)d^{k-c}\right) \leq \frac{n}{\tilde{k}} \sum_{i=1}^{\infty} \tilde{k}^{-i} \leq \frac{n}{\tilde{k}}\,.$$

This means that

$$\mathbb{P}\left\{Y_k > n\exp(-d^{k-c})\right\} \leq \mathbb{P}\left\{\bigcup_{i=k+1}^{K} A_i^c\right\} \leq \sum_{i=k+1}^{K} \mathbb{P}\left\{A_i^c\right\} = O(K/n^2) = o(1/n)\,,$$

because, by Theorem 4.1, $\mathbb{P}\left\{A_i^c\right\} = O(1/n^2)$. ☐

Throughout this thesis, we have ignored the time needed to compute the hashing values which is basically because we often deal with only two hash functions. However, since in this chapter we are going to consider the case of increasing the choices for any $d \geq 2$, we change our assumption temporarily. Throughout this chapter only, we assume that any hashing value can be computed in one unit of time; and any ball can be accessed and examined in one time unit. Furthermore, we assume that $\alpha := m/n$ is constant, and $d = o(\log n)$.

4.1 Increasing the Choices

We have seen in the aftermath of Theorem 2.6 that one can dramatically decrease the maximum bin load by slightly increasing the number of bins. More precisely, Theorem 2.6 states that for any integer $r \in [2, \log_2 n/4]$, the maximum bin load of Uniform-GreedyMC($\lfloor n^{1+1/r} \rfloor$, n), which inserts n balls into $\lfloor n^{1+1/r} \rfloor$ bins, is at most $\log_2 r + \Theta(1)$, w.h.p. Therefore, if we take $r = \log\log n$, the maximum search time of Uniform-ShortChain($\lfloor n^{1+1/r} \rfloor, n$) is at most $2\log_2 \log\log n + \Theta(1)$, w.h.p.

On the other hand, one might also think about increasing the number of choices available for each ball or key. Recall that upon termination of algorithm Uniform-GreedyMC(n, m, d), Theorem 0.2 asserts that the maximum bin load is $\log_d \log n \pm \Theta(1)$, asymptotically almost surely. These bounds are true for any integer $d \geq 2$, even if it is a function on n. In particular, if we choose $d = \Theta(\log\log n/\log\log\log n)$, then the maximum bin load is $\Theta(\log\log n/\log\log\log n)$, w.h.p. This is a useful improvement for many applications such as on-line load balancing and dynamic resource allocation. The trade-off, however, is that the total allocation (insertion) time per ball increases to $\Theta(\log\log n/\log\log\log n)$ as opposed to $\Theta(1)$ when d is a constant. On the other hand, increasing the number of choices does not provide a big help for applications like hashing.

Three-way Chaining

In hashing with chaining, the emphasis is on the average and the worst-case search times. The worst-case search time is related to the time needed to search the d longest chains in the hash table, (plus d, for computing the d hash functions). The next theorem shows that the worst-case search time of Uniform-ShortChain(n, m, d) is $d\log_d \log n \pm \Theta(d)$, w.h.p. Therefore, allowing three choices for each element—because the minimum of $d/\log d$ occurs when $d = 3$—suffices to obtain the optimal worst-case search time of Uniform-ShortChain(n, m, d).

Theorem 4.2. *Let $n, m, d \in \mathbb{N}$ such that $2 \leq d = o(\log n)$ and $m = \Theta(n)$. Upon termination of algorithm* Uniform-ShortChain(n, m, d)*, the worst-case search time is $d\log_d \log n \pm \Theta(d)$, w.h.p.*

Proof. It is easy to see that the maximum search time is $d\log_d \log n + O(d)$, because the longest linked list in the hash table is at most $\log_d \log n + O(1)$, w.h.p., and in the worst-case we have to search d distinct lists. For the lower bound, we know that w.h.p., the longest linked list is at least $\xi_n := \log_d \log n - c$, for some positive constant c. This means that w.h.p., there is a key b at level ξ_n; that is, the d choices of b are chains of length at least $\xi_n - 1$. However, since $d = o(\log n)$, then by the birthday paradox, the d choices of b are distinct, w.h.p. Thence, the maximum search time is at least $d + d(\xi_n - 1) = d\xi_n$, w.h.p. □

This means that w.h.p., the maximum search time is $2.885390... \times \log\log n \pm \Theta(1)$, when $d = 2$, and it is $2.730717... \times \log\log n \pm \Theta(1)$, when $d = 3$. However, we know that by Theorem 1.1, the average search time of UNIFORM-SHORTCHAIN, when $d = 2$, is at most twice the average search time of the uniform classical algorithm CLASSICCHAIN. Using three hash functions, however, increases the average search time to three times of the classical one. This is because we have to search three chains, and we know that the expected chain length cannot be less than the load factor α. Therefore, one has to compromise between decreasing the maximum search time by a small constant factor versus tripling the average search time. This is the main reason why we opted to limit our study to two-way chaining.

Analogously, the same speedups and trade-offs can be applied for Vöcking's algorithm LEFTMC(n, m, d). Theorem 0.5 states that the maximum bin load upon termination of LEFTMC(n, m, d) is $\log\log n/(d \log \phi_d) + \Theta(1)$, w.h.p. That is, if we use $d = \Theta(\sqrt{\log\log n})$ hash functions, the maximum bin load reduces to $\Theta(\sqrt{\log\log n})$, because $d \log \phi_d > (d-1)\log 2$. The benefits of this reduction can be seen in load balancing, resource allocation problems and even in hashing. However, the insertion time increases to $O(d)$. The following theorem follows trivially.

Theorem 4.3. *Let $n, m, d \in \mathbb{N}$ such that $2 \leq d = o(\log n)$ and $m = \Theta(n)$. Upon termination of algorithm* LEFT-SHORTCHAIN(n, m, d)*, the worst-case search time is*

$\log\log n/\log\phi_d \pm \Theta(d)$, *w.h.p. So if* $d = o(\log\log n)$, *then the worst-case search time is asymptotic to* $\log\log n/\log\phi_d$, *in probability.*

Notice that this is an improvement on the wosrt-case performance of UNIFORM-SHORTCHAIN, because for any $d \geq 2$,

$$\frac{3}{\log 3} = 2.730717... > 2.0998... = \frac{1}{\log 1.61} > \frac{1}{\log \phi_d} > \frac{1}{\log 2} = 1.44269...\,,$$

because $1.61... = \phi_2 < \phi_3 < \phi_4 < \cdots < 2$, and $\lim_{d\to\infty}\phi_d = 2$.

4.2 Hashing with Balanced Trees

One can speedup two-way chaining algorithm by inserting the data within each cell in a balanced binary search tree such as AVL tree, a red-black tree, or a B-tree. Instead of a linked list, suppose, for instance, that each cell in the hash table points to a separate AVL tree, and with each cell, we save the size of the tree it points to. As usual, two independent truly uniform hash functions are used, then, to choose two AVL trees for each element, and we insert the element into the smallest one, breaking ties arbitrarily. As a shortcut, we write AVL-HASH(n, m) to refer to this method of hashing.

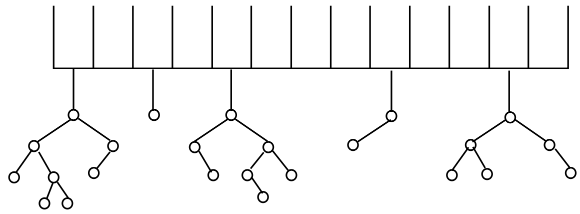

Figure 4.1: Hashing with AVL binary search trees

It is known [103, §6.2.3] that the height of an AVL binary search tree of size n is at most $\log_\phi n + \Theta(1)$, where ϕ is the golden ratio $(1+\sqrt{5})/2 = 1.61803398...$, and the insertion and deletion time is $O(\log n)$. This implies that the maximum search time in AVL-HASH(n, m) is $2\log_\phi \log\log n + \Theta(1)$, w.h.p.; and the worst-case insertion time is $O(\log\log\log n)$, w.h.p. Nonetheless, the following theorem asserts that the average expected insertion time in AVL-HASH(n, m) is constant.

Theorem 4.4. *Let $m, n \in \mathbb{N}$ such that $m/n \geq 1$ is constant. The average insertion time in* AVL-HASH(n, m) *is constant, w.h.p., and hence, the average expected insertion time is constant.*

Proof. Let T be the total insertion time in AVL-HASH(n, m). It is obvious that $m \leq T < m^2$. For $k \in [m]$, let N_k be the number of AVL trees in AVL-HASH(n, m) of size exactly $\lfloor k + m/n \rfloor$, which is less than the number of bins in UNIFORM-GREEDYMC(n, m) of load at least $\lfloor k + m/n \rfloor$. Let A denote the event that $N_k \leq n\exp(-2^{k-c})$, for all $k \in (c, m]$, where c is the same constant in Theorem 4.1. Let N_0 be the number of AVL trees of size at most $m/n + c$. For any AVL tree of size k, we bound the total insertion time of its elements by k^2. Now notice that if A is true, then

$$\begin{aligned} T &\leq (m/n + c)^2 N_0 + \sum_{k=\lceil c \rceil}^{m} (m/n + k)^2 N_k \\ &\leq (m/n + c)^2 n + \sum_{k=\lceil c \rceil}^{m} (m/n + k)^2\, n\exp(-2^{k-c}) \leq \gamma n\,, \end{aligned}$$

for some constant γ. In other words, the event $[\,T > \gamma n\,]$ implies A^c. Theorem 4.1 says that $\mathbb{P}\left\{N_k > n\exp(-2^{k-c})\right\} = O(1/n^2)$, for all $k \in (c, m]$, and thus, the union bound leads to $\mathbb{P}\{T > \gamma n\} \leq \mathbb{P}\{A^c\} = O(1/n)$. Hence,

$$\mathbf{E}[\,T\,] = \sum_{i=1}^{n^2} \mathbb{P}\{T \geq i\} \leq \gamma n + n^2\mathbb{P}\{T > \gamma n\} = O(n)\,.$$

Thus, $\mathbf{E}[\,T\,]/m$ is constant, and T/m is constant, w.h.p. $\square$

4.3 Partially Off-line Processes

The memory algorithm UNIFORM-GREEDYMC(n, n) uses during the insertion of any ball is $O(1)$, as upon the arrival of each ball it only needs to choose two bins, compare their loads, and then insert the ball into the least loaded bin. On the other hand, the off-line version of UNIFORM-GREEDYMC(n, n) uses $\Theta(n)$ memory, because it waits for the n balls to arrive, and finds an assignment after knowing all the $2n$ choices of the balls. The off-line process has more knowledge, and hence more freedom and power, because it is not obliged to follow the greedy paradigm of inserting each ball into the least full bin. The on-line process achieves $\log_2 \log n + \Theta(1)$ maximum bin load, w.h.p.; while in the off-line version, one can find an assignment that decreases the maximum bin load to two, w.h.p. So at the cost of larger memory and longer waiting time, one can decrease the maximum bin load.

This promotes the idea of partially off-line UNIFORM-GREEDYMC with memory of size $k = o(n)$. Keeping in mind that each ball chooses two bins independently and uniformly at random, with replacement, suppose we divide the allocation process into stages. At each stage we wait for $k = o(n)$ balls to arrive—except possibly the last stage—and then we assign them off-line to the bins (via algorithm ASSIGN, described below) such that w.h.p., each k balls are inserted into k distinct bins. We write STAGESMC(n, n, k) to refer to this partially off-line multiple -choice allocation process with memory of size k. We would like to study the maximum bin load of STAGESMC(n, n, k) for different ranges of k.

The off-line algorithm ASSIGN inserts each group of k balls into the bins so that w.h.p., each ball ends up in a distinct bin. The input of the algorithm is the k balls and their choices of bins, where each ball has two bins. For each bin u, let $\psi(u)$ be the number of times the bin is chosen, and suppose that $\psi(u)$ is saved with each bin and is kept updated, and that $\psi(u)$ can be accessed in one time unit. We also assume that there are two-way links between each ball and its two chosen bins.

ASSIGN:

Let L be the list of all bins u such that $\psi(u) = 1$.

for each bin $u \in L$ **do**

Let b be the ball that has chosen u, and let v be the second bin chosen by b.

Insert b into u, remove it from L, and let $\psi(v) \leftarrow \psi(v) - 1$.

If $\psi(v) = 1$, **then** add v to the list L.

If $\psi(v) = 0$, **then** remove v from the list L.

end for

Insert all balls that are not inserted yet into their first-choice bins.

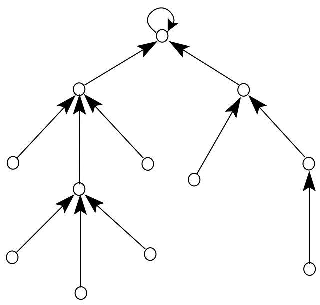

Figure 4.2: An illustration of the algorithm ASSIGN.

To understand the algorithm, recall the random graph $\mathbb{G}(n, k)$ which is based on the off-line version of UNIFORM-GREEDYMC(n, k). The k edges in $\mathbb{G}(n, k)$, which may contain loops and multiedges, are constructed from the $2k$ random choices of the k balls. Notice that for any $k = o(n)$, the random graph $\mathbb{G}(n, k)$ is asymptotically equivalent to the classical Erdös and Rényi's random graph $G(n, k)$ which is known to be a forest, w.h.p.; indeed, the probability that the random graph $\mathbb{G}(n, k)$ contains a cycle is not more than

$$\sum_{i=1}^{k} \binom{n}{i} \mathbb{P}\left\{\mathrm{Bin}(k, i^2/n^2) \geq i\right\} \leq \sum_{i=1}^{k} \left(\frac{en}{i}\right)^i \left(\frac{eki}{n^2}\right)^i \leq \sum_{i=1}^{k} \left(\frac{e^2 k}{n}\right)^i \leq \frac{2e^2 k}{n} = o(1)\,,$$

for n large enough, where we have used the binomial inequality (3) of Lemma 0.1. This means that the $\mathbb{G}(n,k)$ can be oriented such that the maximum out-degree is one, i.e., a 1-orientation can be found. In fact, a 1-orientation can be achieved even if each tree has at most one loop. This can be done as follows. For each vertex of degree one (a leaf) in any tree, orient its incident edge outward, i.e., toward the parent. Now ignore the oriented edges and repeat the same step for the remaining undirected trees. The last edge to be oriented in any tree is the loop, if the tree has one. This is what the algorithm Assign is in essence doing, equivalently. The running time of the algorithm is $O(k)$.

Now let us see how the maximum bin load changes with k. First of all, by the birthday paradox, we know that if $k = o(\sqrt{n})$, then w.h.p., the $2k$ choices of the balls do not overlap, and hence each ball ends up in a distinct bin [105]. That is, stochastically, there is no difference between the off-line and on-line insertion of the k balls. Thus, in this case, StagesMC(n,n,k) is not better than the on-line version, as it is confirmed by the second part of Theorem 0.2.

To study the maximum bin load for general k, we need to consider the analogous problem in the classical allocation process Classical(n,n). Recall that the maximum bin load of Classical(n,n), where each ball is inserted into a bin chosen independently and uniformly at random with replacement, is asymptotic to $\xi_n := \log n / \log\log n$, in probability. For simplicity, assume that $n = kr$, for some positive integers $k = o(n)$ and r. Now consider the following adaptive process which is divided into r stages. At each stage we wait for k balls to arrive, and then we assign them to k distinct bins chosen at random, where any k distinct bins are equally likely to be chosen, i.e., the probability that we choose any k distinct bins is $1/\binom{n}{k}$. Let us refer to this process by ClassicStages(n,n,k). The question now is: does this yield a better maximum bin load than ξ_n? Evidently, the answer depends on k.

Again, if $k = o(\sqrt{n})$, then by the birthday paradox, the modified process is equivalent to the classical allocation process, and so the maximum bin load does not improve. Indeed, the following theorem asserts that if $k \geq cn/\xi_n$, for some constant $c > 1$, the maximum bin load w.h.p., decreases to n/k which is strictly less than ξ_n; otherwise, the maximum bin load does not change, i.e., it is asymptotic to ξ_n, in probability. Before we prove this result we need a brief introduction to it.

Recall that in the classical allocation process $\textsc{Classical}(n, n)$, the distribution of the bin loads $(X_1, \ldots, X_n)$ is a multinomial with parameters n and $(1/n, \ldots, 1/n)$, where the random variable $X_i \stackrel{\mathcal{L}}{=} \mathrm{Bin}(n, 1/n)$ is the load of the i-th bin. Notice that the X_i are dependent, because the $\sum_{i=1}^{n} X_i = n$; but more importantly, they are negatively associated (or negatively correlated). This, plainly, means that

$$\mathbb{P}\{X_1 < x_1 \,|\, X_2 < x_2\} \leq \mathbb{P}\{X_1 < x_1\} \ ,$$

because if the number of balls decreases in one bin, it is more likely to increase in the other. Therefore, Mallows' inequality, which is a useful tool for proving the lower bound on the maximum bin load, holds here.

Returning back to $\textsc{ClassicStages}(n, n = kr, k)$, let N_i, for $i \in [n]$, denote the load of the i-th bin. Notice that during any stage, the probability that a ball falls into a certain bin is $k\binom{n-1}{k-1}/\binom{n}{k} = k/n = 1/r$. Hence, the load of the i-th bin $N_i \stackrel{\mathcal{L}}{=} \mathrm{Bin}(r,\, 1/r)$. It is evident, however, that the bin load vector $(N_1, \ldots, N_n)$ is not multinomial. So, we cannot conclude directly that the N_i are negatively associated, and use the Mallows inequality. Nonetheless, the inequality

$$\mathbb{P}\{N_1 < t_1 \,|\, N_2 < t_2\} \leq \mathbb{P}\{N_1 < t_1\}$$

still holds for the same reason mentioned above, roughly speaking. In fact, Dubhashi et al. [56, Sec. 5.2] and [57], in the context of the Fermi-Dirac model, proved that the N_i are negatively associated. Hence, we can still use Mallow's inequality.

Theorem 4.5. *Assume that $n = rk$, for some positive integers r and k. Suppose we insert n balls into n bins via* ClassicStages(n, n, k). *Let Y be the maximum bin load upon termination. If $r \leq (1-\epsilon)\log n/\log\log n$, for some constant $\epsilon \in (0,1)$, then $Y = r$, w.h.p.; otherwise, $Y \sim \log n/\log\log n$, in probability.*

Proof. Recall that $N_i \stackrel{\mathcal{L}}{=} \text{Bin}(r, 1/r)$, for $i \in [n]$. Let $\lambda \in [0, r)$ be any integer. Observe that $(1-1/r)^\lambda \geq 1 - \lambda/r$. Thus,

$$\binom{r}{\lambda}\frac{1}{r^\lambda} \;=\; \frac{r(r-1)^\lambda}{(r-\lambda)r^\lambda}\binom{r-1}{\lambda}\frac{1}{(r-1)^\lambda} \;\geq\; \binom{r-1}{\lambda}\frac{1}{(r-1)^\lambda}\,.$$

Thence, if $r \geq t := \lfloor (1-o(1))\log n/\log\log n \rfloor$, we have $\binom{r}{\lambda}/r^\lambda \geq \binom{t}{\lambda}/t^\lambda$. Notice that the sequence $(1-1/r)^r$ increases to $1/e$, monotonically. Thus, for all $i \in [n]$, we have

$$\mathbb{P}\{N_i = \lambda\} \geq \binom{r}{\lambda}\frac{1}{r^\lambda}(1-1/r)^r \geq \binom{t}{\lambda}\frac{1}{t^\lambda}\frac{1}{3} \geq \frac{1}{3t^\lambda}\,,$$

for n large enough. Since the N_i are negatively associated, we have

$$\begin{aligned}\mathbb{P}\{Y < \lambda\} \;&=\; \mathbb{P}\{N_1 < \lambda, \ldots, N_n < \lambda\} \;\leq\; \prod_{i=1}^{n}\mathbb{P}\{N_i < \lambda\}\\ &=\; \prod_{i=1}^{n}(1-\mathbb{P}\{N_i \geq \lambda\}) \;\leq\; \prod_{i=1}^{n}(1-\mathbb{P}\{N_i = \lambda\})\\ &=\; \exp\left(-n/(3t^\lambda)\right) \;=\; \exp\left(-n^\epsilon\right) \;=\; o(1)\,,\end{aligned}$$

if $\lambda = (1-\epsilon)\log n/\log\log n$, for any constant $\epsilon \in (0,1)$. For the upper bound, the union bound and inequality (3) of Lemma 0.1 yield that

$$\mathbb{P}\{Y > \lambda\} \leq \sum_{i=1}^{n}\mathbb{P}\{N_i > \lambda\} \leq n\left(\frac{e}{\lambda}\right)^\lambda = n^{-\epsilon+o(1)} = o(1)\,,$$

if $\lambda = (1+\epsilon)\log n/\log\log n$, for any constant $\epsilon \in (0,1)$. On the other hand, if $r \leq (1-\epsilon)\log n/\log\log n$, for some constant $\epsilon \in (0,1)$, then $\mathbb{P}\{N_i = r\} = r^{-r} \geq n^{1-\epsilon}$; and so again

$$\mathbb{P}\{Y < r\} \leq \prod_{i=1}^{n}(1-\mathbb{P}\{N_i = r\}) = \exp\left(-n^\epsilon\right) = o(1)\,.$$

Therefore, $Y = r$, w.h.p. $\square$

Now recall that at each stage of StagesMC(n, n, k), where $k = o(n)$, the balls are inserted into k distinct bins, w.h.p. The maximum bin load is at most $\lceil n/k \rceil$, deterministically. However, the important point here is that the k distinct bins may not be uniformly distributed as in ClassicStages(n, n, k). Intuitively, this means that StagesMC(n, n, k) is at most as good as ClassicStages(n, n, k). That is, the maximum bin load of StagesMC(n, n, k) is not better than the maximum bin load of ClassicStages(n, n, k). Thus, if we want the maximum bin load of StagesMC(n, n, k) to be less than $\log_2 \log n$, then, by Theorem 4.5, k must be at least $n/\log_2 \log n$. Thus, if we take $k = \lfloor n/\log\log\log n \rfloor$, then the worst-case search time of StagesMC(n, n, k) is $2\log\log\log n + 2$, w.h.p., and the amortized allocation time is $O(1)$. However, the trade-off is that the allocation time per ball is $O(k)$ which might be an expensive cost.

Corollary 4.2. *Let $n, k \in \mathbb{N}$. If $n/\log_2 \log n \leq k \ll n$, then the maximum bin load of* StagesMC(n, n, k) *is $\lceil n/k \rceil$, w.h.p.*

Clearly, this is an improvement for hashing, if the concentration is on the worst-case search time and not the worst-case insertion time. However, this is certainly not helpful for an application where the maximum search time is not an important measure such as on-line load balancing and dynamic resource allocations. Remember that for such applications, we can decrease the maximum bin load of UniformGreedyMC(n, n) by using $d = o(\log\log n)$ hash functions instead of only two, where the allocation time for each ball (and the amortized allocation time) is $O(d)$. The worst-case search time resulting from this speedup, however, is not plausible.

4.4 Processes with Load Thresholds

One can improve the maximum bin load of Uniform-GreedyMC(n, m) by reassigning the balls that are above certain level, say $\lfloor L_n + m/n \rfloor$, assuming that the reassignment can be done efficiently in some sense. This depends, trivially, on the number of balls we have to reassign. Given that $L_n = o(\log\log n)$, and $L_n \xrightarrow{n} \infty$, Corollary 4.1 says that w.h.p., the number of "bad balls" that exceed the $\lfloor L_n + m/n \rfloor$ load threshold in Uniform-GreedyMC(n, m) is at most $n \exp(-2^{L_n - c}) = o(n)$. For instance, if $L_n = \log_2 \log\log n + c$, the number of balls above the level $\lfloor L_n + m/n \rfloor$ is at most $n/\log n = o(n)$. Now since the number of bad balls is "small", one can rearrange these balls off-line in a special way so that each ball ends up in a distinct bin. This has been demonstrated above by the off-line algorithm Assign, which is based on the fact that the random graph generated from the ball choices is a forest, w.h.p.

This leads to the following partially off-line Uniform-GreedyMC. First we choose our load threshold L_n so that $L_n = o(\log\log n)$, and $L_n \xrightarrow{n} \infty$. We divide the allocation process into two mini-processes. The first one follows the same greedy strategy of the on-line algorithm Uniform-GreedyMC (assigning the balls upon arrival to the least full bin among two bins chosen independently and uniformly at random, breaking ties at random), but with only one exception. Any ball that chooses two bins with loads more than $\lfloor L_n + m/n \rfloor$ is not inserted, and put aside for the second mini-process to deal with. Notice that the number of balls that are put aside is at most $o(n)$, w.h.p. Hence, once all the m balls arrive, the second mini-process uses the off-line algorithm Assign to insert the remaining balls. Let us refer to this partially off-line process by PO-Threshold(n, m, L_n).

The maximum bin load of PO-Threshold(n, m, L_n) is $\lfloor L_n + m/n \rfloor + 1$, w.h.p., and hence the worst-case search time is $2 \lfloor L_n + m/n \rfloor + 4$, w.h.p. The allocation

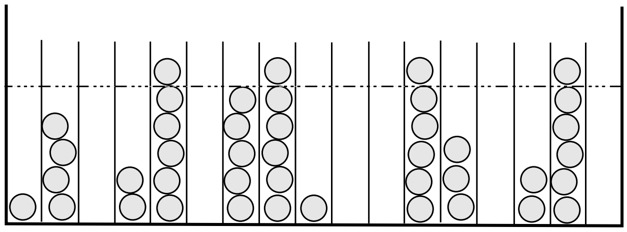

Figure 4.3: An illustration of the process PO-Threshold(n, m, L_n).

time is deterministically constant for each ball inserted during the first phase, and it is $t_n := \Theta(n \exp(-2^{L_n \pm \Theta(1)}))$, w.h.p., for all the balls inserted during the second phase. Thus, the amortized allocation time is still $O(1)$; and, obviously, the worst-case allocation time is t_n, w.h.p. But what about the expected allocation time? Unfortunately, even the expected allocation time is at least t_n, as it shown in this lemma.

Lemma 4.1. *Let $n, m, L_n \in \mathbb{N}$ such that $m/n \geq 1$ is constant, and $1 \ll L_n \ll \log\log n$. In algorithm* PO-Threshold(n, m, L_n)*, the expected allocation time of a certain ball picked uniformly at random is t_n.*

Proof. Let T be the allocation time of a certain ball picked uniformly at random. Let A be the event that the number of balls inserted during the second phase is at least $\zeta_n := n \exp(-2^{L_n+b+1})$, where b is the same constant in Theorem 4.1. We know that A occurs w.h.p. Let B be the event of being inserted during the second phase. Recall that the running time of the algorithm Assign is $O(k)$ for assigning k balls. By Markov's inequality, and for n large enough, we have

$$\begin{aligned}
\mathbf{E}[T] &\geq \zeta_n \mathbb{P}\{T \geq \zeta_n\} \geq \zeta_n \mathbb{P}\{A \cap B\} \geq \zeta_n \mathbb{P}\{A\} \mathbb{P}\{B \mid A\} \\
&\geq \frac{\zeta_n}{2} \frac{n}{m} \exp\left(-2^{L_n+b+1}\right) = \Omega\left(n \exp\left(-2^{L_n+b+2}\right)\right) .
\end{aligned}$$

Next, let C be the event that the number of balls inserted during the second phase is at most $\xi_n := n \exp(-2^{L_n - c})$, where c is the same constant in Theorem 4.1. Notice that

$$\begin{aligned} T &\le T\,\mathbb{I}_{[\,B\cap C\,]} + T\,\mathbb{I}_{[\,B^c\,]} + T\,\mathbb{I}_{[\,C^c\,]} \\ &\le O(\xi_n) + O(1) + m\,\mathbb{I}_{[\,C^c\,]}\ , \end{aligned}$$

However, Corollary 4.1 says that $\mathbb{P}\{C^c\} = o(1/n)$, and so $\mathbf{E}[\,T\,] = O(\xi_n) + o(1)$. □

Once again, because of the high cost of the allocation time during the second off-line mini-process, the process PO-Threshold may not be the best choice for on-line load balancing and dynamic resource allocation.

Uniform-GreedyMC(n, m, d)	**where $d \geq 2$**
maximum bin load	$= \log_d \log n + O(1)$, w.h.p.
maximum search time	$= d \log_d \log n + O(d)$, w.h.p.
maximum allocation time	$= \Theta(d)$
amortized allocation time	$= \Theta(d)$
avg. exp. allocation time	$= \Theta(d)$
Uniform-GreedyMC$(\lfloor n^{1+1/r} \rfloor, m)$,	**where $2 \leq r \leq \log_2 n/4$**
maximum bin load	$\leq \log_2 r + \Theta(1)$, w.h.p.
maximum search time	$\leq 2\log_2 r + \Theta(1)$, w.h.p.
maximum allocation time	$= \Theta(1)$
amortized allocation time	$= \Theta(1)$
avg. exp. allocation time	$= \Theta(1)$
AVL-Hash(n, m),	**where ϕ is the golden ratio.**
maximum bin load	$= \log_\phi \log\log n + \Theta(1)$, w.h.p.
maximum search time	$= 2\log_\phi \log\log n + \Theta(1)$, w.h.p.
maximum allocation time	$= \Theta(\log\log\log n)$, w.h.p.
amortized allocation time	$= O(\log\log\log n)$ (worst-case)
avg. exp. allocation time	$= \Theta(1)$
StagesMC(n, m, k_n),	**where $n/\log_2 \log n \leq k_n \ll n$**
maximum bin load	$(=) \lceil m/k_n \rceil$, w.h.p.
maximum search time	$(=) 2\lceil m/k_n \rceil + 2$, w.h.p.
maximum allocation time	$= \Theta(k_n)$
amortized allocation time	$= \Theta(1)$
avg. exp. allocation time	$= \Theta(k_n)$
PO-Threshold(n, m, L_n),	**where $1 \ll L_n \ll \log\log n$**
maximum bin load	$= \lfloor L_n + m/n \rfloor + 1$, w.h.p.
maximum search time	$= 2\lfloor L_n + m/n \rfloor + 4$, w.h.p.
maximum allocation time	$= \Theta\left(n \exp\left(-2^{L_n \pm \Theta(1)}\right)\right)$, w.h.p.
amortized allocation time	$= \Theta(1)$
avg. exp. allocation time	$= \Theta\left(n \exp\left(-2^{L_n \pm \Theta(1)}\right)\right)$

Table 4.1: The performances of the above processes, where m/n is constant.

Part II

Hashing with Open Addressing

Open addressing is a collision resolution method that does not use chains or pointers. The idea of hashing with open addressing appears to have been suggested, first, around 1953 by G. M. Amdahl, E. M. Boehme, N. Rochester, and A. L. Samuel who also used, for the first time, linear probing [103, p. 547]. The first published article about open addressing with linear probing is the Russian monograph written by Ershov [62]. About the same time, Peterson [147] wrote the first major paper that analyzes the average performance of uniform probing. Statistical data about the behavior of linear probing were also given. Knuth [102, 103] reported that he analyzed the average performance of linear probing in unpublished notes in 1963. The first published analysis is done by Konheim and Weiss [106]. Morris [136] introduced random probing.

In this part of the thesis, we focus on some open addressing ideas inspired by the two-way chaining paradigm. In particular, we study the concept of two-way linear probing. In Chapter 5, we recall some of the related history, and we analyze the basic idea of two-way linear probing demonstrating that it is not always fruitful. Subsequently, we introduce, in Chapter 6, some successful two-way linear probing algorithms that improve the performance.

Chapter 5

Two-way Linear Probing: the Naked Idea

We study on-line open addressing schemes that use two linear probe sequences to find possible hashing cells for the keys as follows. Each key chooses two initial cells (from a hash table with n cells) independently and uniformly ar random, with replacement. From each initial cell, we probe linearly, and cyclically whenever the last cell in the table is reached, to find two empty cells which we call terminal cells. The key then is inserted into one of these terminal cells according to a fixed strategy. We consider strategies that utilize the greedy multiple-choice paradigm. For example, one of the trivial strategies inserts each key into the terminal cell found by the shortest probe sequence. Another simple strategy inserts each key into the terminal cell that is adjacent to the smallest cluster, where a cluster is an isolated set of consecutively occupied cells. Unfortunately, the performances of these two strategies are not as good as we might expect. We prove that the maximum unsuccessful search time is $\Omega(\log n/\log\log n)$, w.h.p., when any of these two strategies is used to construct a hash table with constant load factor. We also show that an $\Omega(\log\log n)$ universal lower bound holds for any strategy that uses two linear probe sequences, even if the initial cells are chosen according to arbitrary probability distributions.

5.1 History and Motivation

In classical open addressing hashing [147], m keys are hashed sequentially and on-line into a table of size $n > m$, (that is, a one-dimensional array with n cells which we denote by the set $\mathcal{T} = \{0, \ldots, n-1\}$), where each cell can harbor at most one key. Each key x has only one infinite *probe sequence* $f_i(x) \in \mathcal{T}$, for $i \in \mathbb{N}$, where $f_i(x)$ is the i-th probe available for the key x. During the insertion process, if a key is mapped to a cell that is already occupied by another key, a collision occurs, and another probe is required. The probing continues until an empty cell is reached where a key is placed. For further details see [103, 80, 169]. This method of hashing is pointer-free, unlike hashing with separate chaining which we studied in the first part of the thesis.

Probing and Replacement

Open addressing schemes are determined by the type of probe sequence, and the replacement strategy for resolving collisions. Some of the commonly used probe sequences are:

1. **Random Probing [136]:** For every key x, the infinite sequence $f_i(x)$ is assumed to be independent and uniformly distributed over $\mathcal{T}$. That is, we require to have an infinite sequence f_i of truly uniform and independent hash functions. If for each key x, the first n probes of the sequence $f_i(x)$ are distinct, i.e., it is a random permutation, then it is called uniform probing [147].

2. **Linear Probing [147]:** For every key x, the first probe $f_1(x)$ is assumed to be

uniform on $\mathcal{T}$, and the next probes are defined by $f_{i+1}(x) = f_i(x) + 1 \mod n$, for $i \in [n]$. So we only require f_1 to be a truly uniform hash function.

3. **Double Probing [15]:** For every key x, the first probe is $f_1(x)$, and the next probes are defined by $f_{i+1}(x) = f_i(x) + g(x) \mod n$, for $i \in \mathbb{N}$, where f_1 and g are truly uniform and independent hash functions.

Random and uniform probings are, in some sense, the idealized models [164, 178], and their plausible performances are among the easiest to analyze; but obviously they are unrealistic. Linear probing is perhaps the simplest to implement, but it behaves badly when the table is almost full. Double probing can be seen as a compromise.

During the insertion process of a key x, suppose that we arrive at the cell $f_i(x)$ which is already occupied by another previously inserted key y, that is, $f_i(x) = f_j(y)$, for some $j \in \mathbb{N}$. Then a replacement strategy for resolving the collision is needed. Three strategies have been suggested in the literature (see [138] for other methods):

1. FIRST COME FIRST SERVED (FCFS) [147]: The key y is kept in its cell, and the key x is referred to the next cell $f_{i+1}(x)$.

2. LAST COME FIRST SERVED (LCFS) [151]: The key x is inserted into the cell $f_i(x)$, and the key y is pushed along to the next cell in its probe sequence, $f_{j+1}(y)$.

3. ROBIN HOOD [29, 28]: The key which travelled the furthest is inserted into the cell. That is, if $i > j$, then the key x is inserted into the cell $f_i(x)$, and the key y is pushed along to the next cell $f_{j+1}(y)$; otherwise, y is kept in its cell, and the key x tries its next cell $f_{i+1}(x)$.

Average Performance

Evidently, the performance of any open addressing scheme deteriorates when the load factor $\alpha := m/n$ of the hash table approaches 1, as the cluster sizes increase, where a *cluster* is an isolated set of consecutively occupied cells (cyclically defined) that are bounded by empty cells. Therefore, we shall assume, throughout this chapter, that $\alpha \in (0, 1)$ is a constant. The asymptotic average-case performance has been extensively analyzed for random and uniform probing [147, 136, 164, 109, 178, 21], linear probing [102, 103, 106, 125], and double probing [15, 84, 115, 162, 158]. The expected search times were proven to be constants, more or less, depending on α only. Recent results about the average-case performance of linear probing, and the limit distribution of the construction time have appeared in [166, 67, 104]. See also [78, 148, 4] for the average-case analysis of linear probing for nonuniform hash functions.

It is worth noting that the average search time of linear probing is independent of the replacement strategy; see [147, 103]. This is because the insertion of any order of the keys results in the same set of occupied cells, i.e., the cluster sizes are the same; and hence, the total displacement of the keys—from their initial hashing locations—remains unchanged. It is not difficult to see that this independence is also true for random and double probings. That is, the replacement strategy does not have any effect on the average successful search time in any of the above probings. In addition, since in linear probing the maximal unsuccessful search time is related to the cluster sizes (unlike random and double probings), the maximum unsuccessful search times in linear probing is invariant to the replacement strategy.

It is known that LCFS [151, 152] and ROBIN HOOD [29, 28, 138, 166] strategies minimize the variance of displacement. Recently, Janson [95] and Viola [165] studied the effect of these replacement strategies on the individual search times in linear probing hashing.

Worst-case Performance

The focal point of this chapter is the worst-case search time which is proportional to the length of the longest probe sequence over all keys (LLPS, for short).

The worst-case performance of linear probing with FCFS policy was analyzed by Pittel [149]. He showed that the maximum cluster size, and hence the LLPS needed to insert (or search for) a key, is asymptotic to $(\alpha - 1 - \log\alpha)^{-1}\log n$, in probability. This bound holds for linear probing with *any* replacement strategy. Chassaing and Louchard [30] studied the threshold of emergence of a giant cluster in linear probing. They showed that when the number of keys $m = n - \omega(\sqrt{n})$, the size of the largest cluster is $o(n)$, w.h.p.; however, when $m = n - o(\sqrt{n})$, a giant cluster of size $\Theta(n)$ emerges, w.h.p.

Gonnet [79] proved that with uniform probing and FCFS replacement strategy, the expected LLPS is asymptotic to $\log_{1/\alpha} n - \log_{1/\alpha}\log_{1/\alpha} n + O(1)$. However, Poblete and Munro [151, 152] showed that if random probing is combined with the LCFS policy, then the expected LLPS is at most $(1+o(1))\Gamma^{-1}(\alpha n) = O(\log n/\log\log n)$, where Γ is the gamma function.

On the other hand, the ROBIN HOOD strategy with random probing leads to a more striking performance. Celis [28] first proved that the expected LLPS is $O(\log n)$. However, Devroye, Morin and Viola [45] tightened the bounds and revealed that the LLPS is indeed $\log_2\log n \pm \Theta(1)$, w.h.p., thus achieving a double logarithmic worst-case insertion and search times for the first time in open addressing without using rehashing techniques. Unfortunately, one cannot ignore the unrealistic assumption in random probing about the availability of an infinite collection of independent and truly uniform hash functions. On the other side of the coin, we already know that ROBIN HOOD policy does not affect the maximum unsuccessful search time in linear probing. However, ROBIN HOOD may be promising with other probing methods.

Other Initiatives

Open addressing methods that rely on rearrangements of keys have been studied in [22, 119, 154, 81, 118, 138]. Most importantly is cuckoo hashing, introduced by Pagh and Rodler [146]. The scheme exploits the LCFS replacement policy in a hash table partitioned into two parts. The worst-case search time is at most two, and the amortized expected insertion time is constant. However, the pitfall of this scheme is that it depends on a rehashing process which uses a wealthy source of provably good independent hash functions. See also [44, 69, 53, 141]. Broder and Karlin [23] suggested a multilevel hashing scheme with $O(\log\log n)$ worst-case search time, but it uses $O(\log\log n)$ hash functions and a rehashing technique. Many real-time static and dynamic perfect hashing schemes achieving constant worst-case search time, and linear (in the table size) construction time and space were designed in [74, 23, 48, 52, 51, 50, 142, 144]. Usually such schemes employ more than a constant number of perfect hash functions chosen from an efficient universal class. Some of them even use $O(n)$ functions. For a more detailed account on these schemes see Section 1.1.

5.2 Two-way Linear Probing

Inspired by the two-way chaining paradigm and its powerful performance, we promote the concept of open addressing hashing with two-way linear probing. The essence of the proposed concept is based on the idea of allowing each key to generate two independent linear probe sequences and making the algorithm decide, according to some strategy, at the end of which sequence the key should be inserted. Formally, each input key x chooses two cells independently and uniformly at random, with replacement. We call these cells the *initial hashing cells* available for x. From each initial hashing cell, we start a linear probe sequence (with FCFS policy) to find an empty cell where we stop. Thus, we end up with two unoccupied cells. We call these cells the *terminal hashing cells*.

Definition 5.1. An on-line two-way linear probing algorithm is an open addressing hashing algorithm that inserts keys sequentially into cells using a certain strategy and does the following upon the arrival of each key:

1. It chooses two initial hashing cells independently and uniformly at random, with replacement.

2. Two terminal (empty) cells are then found by linear probe sequences starting from the initial cells.

3. The key is inserted into one of these terminal cells.

The question now is: into which terminal cell should we insert the key x? A two-way linear probing algorithm could follow one of the strategies we mentioned earlier: it may insert the key at the end of the shortest probe sequence, or into the terminal cell that is adjacent to the smallest cluster. Others may make a decision even before linear probing starts. In any of these algorithms, the searching process for any key is basically the same: just start probing in both sequences alternately, until the key is found, or the two empty cells at the end of the sequences are reached in the case of an unsuccessful search. Thus, the maximum unsuccessful search time is at most twice the size of the largest cluster plus two.

In the next section, we prove that the maximum unsuccessful search time of any two-way linear probing algorithm, that satisfies the above definition, is $\Omega(\log\log n)$. Unfortunately, not every two-way linear probing algorithm has a matching upper bound on its worst-case performance. In Section 5.4, we prove that there are classes of two-way linear probing algorithms that behave poorly; in particular, we analyze the two algorithms, described above, and show that the hash table, asymptotically and almost surely, contains a cluster of size $\Omega(\log n/\log\log n)$. However, we present, in Chapter 6, two-way linear probing algorithms that achieve $\Theta(\log\log n)$ worst-case search time.

Two-way linear probing hashing—if used successfully as we are going to see in the next chapter—has several advantages over other proposed hashing methods: it reduces the worst-case behavior of hashing, it requires only two hash functions, it is easy to parallelize, it is pointer-free and easy to implement, and it does not require any rearrangement of keys or rehashing. Its average-case performance can be at most twice the classical linear probing, and its maximum cluster size is $O(\log\log n)$, unlike all other methods. Furthermore, it is not necessary to employ perfectly random hash functions. We believe that hash functions with smaller degree of universality, (e.g., $O(\log n)$-universal), such as the ones in [48, 101, 162, 161, 158, 146] will be sufficient.

Before we embark on the analysis, we should remind the reader that the hashing assumptions stated in Section 0.4 are also applied in this part of the thesis. In particular, we assume the following. We have a set of input keys $\mathcal{K} \subseteq \mathcal{U}$ of size m to be hashed into a hash table $\mathcal{T} = \{0, \ldots, n-1\}$ such that each cell contains at most one key. The process of hashing is sequential and on-line, unless otherwise stated. Furthermore, we assume that the linear probe sequences always move cyclically from left to right of the hash table. The replacement strategy of all of the introduced algorithms is FCFS. The *insertion time* is defined to be the number of probes the algorithm performs to insert a key. Similarly, the *search time* is defined to be the number of probes needed to find a key, or two empty cells in the case of unsuccessful search. Observe that unlike classical linear probing, the insertion time of two-way linear probing is not equal to the successful search time.

5.3 Universal Lower Bound

The following lower bound holds for any two-way linear probing hashing scheme, in particular, the ones that are presented here and in the next chapter.

Theorem 5.1. *Let $n, m \in \mathbb{N}$, and assume that $\alpha := m/n \in (0,1)$ is a constant. Let* **A** *be any on-line two-way linear probing algorithm that inserts m keys into a hash table of size n. Then upon termination of* **A**, *w.h.p., the table contains a cluster of size at least* $\log_2 \log n - \Theta(1)$.

Proof. Imagine that we have a bin associated with each cell in the hash table. Recall that for each key x, algorithm **A** chooses two initial hashing cells, and hence two bins, independently and uniformly at random, with replacement. Algorithm **A**, then, probes linearly to find two (possibly identical) terminal cells, and inserts the key x into one of them. Suppose that after the insertion of each key x, we also insert a ball into the bin associated with the initial cell from which the algorithm started probing to reach the terminal cell into which the key x was placed. If both of the initial cells lead to the same terminal cell, then we break the tie randomly. Clearly, if there is a bin with k balls, then there is a cluster of size of at least k, because the k balls represent k distinct keys that belong to the same cluster. This means that we have an algorithm that inserts m balls into n bins where each ball is placed into a bin among two bins chosen independently and uniformly at random, with replacement. Thus, by the second part of Theorem 0.2, the maximum bin load upon termination of algorithm **A** is at least $\log_2 \log n - \Theta(1)$, w.h.p. ☐

The above lower bound is valid for all algorithms that satisfy Definition 5.1. A more general lower bound can be established on all open addressing schemes that use two linear probe sequences where the initial hashing cells are chosen according to some (not necessarily uniform or independent) probability distributions defined on the cells. We still assume that the probe sequences are used to find two (empty) terminal hashing cells, and the key is inserted into one of them according to some strategy. We call such schemes *nonuniform two-way linear probing*. The proof of the following theorem is similar to Theorem 5.1, but uses Vöcking's lower bound (Theorem 0.5).

Theorem 5.2. *Let $n, m \in \mathbb{N}$, and assume that $\alpha := m/n \in (0,1)$ is a constant. Let* $\mathbf{A}$ *be any nonuniform two-way linear probing algorithm that inserts m keys into a hash table of size n where the initial hashing cells are chosen according to some probability distributions. Then the maximum cluster size produced by* $\mathbf{A}$*, upon termination, is at least* $0.72... \times \log_2 \log n - \Theta(1)$*, w.h.p.*

5.4 Life is not Always Good!

The idea of two-way linear probing alone is not always sufficient to pull off a plausible hashing performance. Indeed, a large group of two-way linear probing algorithms have an $\Omega(\log n / \log\log n)$ lower bound on their worst-case search time. In this section, we characterize some of the two-way linear probing algorithms that behave disappointingly.

Ignorant Algorithms

A two-way linear probing algorithm is not expected to have $O(\log\log n)$ worst-case search time, if its decisions of where to place the keys are made solely from the information obtained by only one linear probing sequence. For instance, consider the following strategy: each key is inserted into the first terminal cell, if it is reached by not more than k probes; otherwise, the key is placed into the second terminal cell. Information about the second terminal cell is basically ignored. Indeed, we may as well postpone selecting the second initial hashing cell, if needed, until after the algorithm makes its decision. We prove next that such algorithms are doomed to have poor performance. Notice that we still use both probe sequences to search for any key by probing linearly and alternatingly, until the key is found, or two empty cells are reached in the case of an unsuccessful search. Thus, the maximum unsuccessful search time is at most twice the size of the largest cluster plus two.

Theorem 5.3. *Let $n, m \in \mathbb{N}$ such that $\alpha := m/n \in (0, 1)$ is a constant. Suppose that we have a two-way linear probing algorithm that inserts m keys into n cells such that the decision of where to insert any key is made before its second initial hashing cell is chosen. In other words, the decisions are made without knowing any information about the second choices. Then the hash table must have a giant cluster of size at least $\lambda(n) := \left\lceil \sqrt{(2-\epsilon)\log n/\log\log n} \right\rceil$, w.h.p., for any constant $\epsilon \in (0, 1)$.*

Proof. Fix $\epsilon \in (0, 1)$. Without loss of generality, we assume that during the insertion process, the second initial cell for any key is chosen if and only if the algorithm decides to insert the key into its second terminal cell. Let S be the set of all keys for which second initial hashing cells are chosen. That is, a key belongs to S if and only if it is inserted into its second terminal cell. For $i \in \mathcal{T} = \{0, \ldots, n-1\}$, let C_i be the number of keys that have chosen the cell i as its second initial hashing cell. Notice that every key that has chosen the cell i as its second initial cell is inserted into the empty cell at the end of the cluster containing cell i. This means that the size of the cluster that contains the cell i is at least C_i. Thus, if $\max_i C_i \geq \lambda$, then the maximum cluster size is at least λ. For $i \in \mathcal{T}$, let F_i be the set of all keys that have chosen the cell i as its first initial hashing cell, and notice that $|F_i| \stackrel{\mathcal{L}}{=} \mathrm{Bin}(m, 1/n)$. Define the set $\mathcal{H} := \{i : |F_i| \geq \lambda\}$. Let A be the event that there is a cell $i \in \mathcal{H}$ such that every key in F_i is inserted into its first terminal hashing cell. For the same reason stated above, if A is true, the maximum cluster size is at least λ. We will show that

$\mathbb{P}\{[\max_i C_i < \lambda] \cap A^c\} = o(1)$. Notice that

$$\mathbb{P}\left\{\left[\max_i C_i < \lambda\right] \cap A^c\right\} \leq \mathbb{P}\left\{\max_i C_i < \lambda \;\middle|\; A^c \cap [\,|\mathcal{H}| \geq N\,]\right\} + \mathbb{P}\{|\mathcal{H}| < N\}\;,$$

where N is defined below. The binomial tail inequality of Lemma 0.2 yields that for n large enough,

$$\mathbf{E}\,[\,|\mathcal{H}|\,] = \sum_{i=0}^{n-1} \mathbb{P}\{|F_i| \geq \lambda\} = n\,\mathbb{P}\{\mathrm{Bin}(m, 1/n) \geq \lambda\} \geq \frac{n}{2e^{\alpha}}\left(\frac{\alpha}{2\lambda}\right)^{\lambda}.$$

Let

$$N := \left\lfloor \frac{n}{4e^{\alpha}}\left(\frac{\alpha}{2\lambda}\right)^{\lambda} \right\rfloor .$$

Clearly, $|\mathcal{H}|$ can be written as a function of m independent random variables, namely, the first initial cells available for the m keys. If one of these initial cells is changed, the random variable $|\mathcal{H}|$ may decrease or increase by at most one. Thus, by McDiarmid's inequality (Lemma 0.3), we see that for n large enough,

$$\mathbb{P}\{|\mathcal{H}| < N\} \leq \mathbb{P}\{|\mathcal{H}| - \mathbf{E}\,[\,|\mathcal{H}|\,] < -N\} \leq \exp\left(\frac{-2N^2}{m}\right) = o(1)\,.$$

Let n be large enough such that

$$\left(1 - \frac{\lambda}{N}\right)^{\lambda} \geq 1 - \frac{\lambda^2}{N} \geq \frac{1}{2}, \quad \text{and} \quad \left(1 - \frac{1}{n}\right)^{n} \geq \frac{1}{3},$$

which can be done because $(1 - 1/n)^n \xrightarrow{n} 1/e$. Then we have

$$\mathbb{P}\{\mathrm{Bin}(N, 1/n) \geq \lambda\} \geq \binom{N}{\lambda}\frac{1}{n^{\lambda}}\left(1 - \frac{1}{n}\right)^{n} \geq \frac{N^{\lambda}(1 - \lambda/N)^{\lambda}}{6\lambda!\,n^{\lambda}} \geq \frac{c^{\lambda^2}}{\lambda^{\lambda^2+\lambda}},$$

for some constant $c \in (0, 1)$. Observe that $|S| = \sum_{i=0}^{n-1} C_i$, and the second initial cells available for all keys in S are independently and uniformly distributed over the hash table. Thus, knowing $|S|$, the vector $(C_1, \ldots, C_n)$ is a multinomial random variable. Also, if A is not true, then the $|S| \geq |\mathcal{H}|$. Consequently, using Mallows' inequality, we obtain

$$\begin{aligned}
\mathbb{P}\left\{\max_i C_i < \lambda \,\middle|\, A^c \cap [\,|\mathcal{H}| \geq N\,]\right\} &\leq \prod_{i=0}^{n-1} \mathbb{P}\left\{C_i < \lambda \mid A^c \cap [\,|\mathcal{H}| \geq N\,]\right\} \\
&\leq \left(1 - \mathbb{P}\left\{\mathrm{Bin}(N, 1/n) \geq \lambda\right\}\right)^n \\
&\leq \exp\left(-\frac{nc^{\lambda^2}}{\lambda^{\lambda^2+\lambda}}\right) = o(1)\,.
\end{aligned}$$

□

Algorithms that Behave Poorly

We consider here the two examples of two-way linear probing algorithms we mentioned earlier. The first algorithm places each key into the terminal cell discovered by the shortest probe sequence. More precisely, once the key chooses its initial hashing cells, we start two linear probe sequences. We proceed, sequentially and alternately, one probe from each sequence until we find an empty (terminal) cell where we insert the key. Formally, let f, $g : \mathcal{U} \to \{0, \ldots, n-1\}$ be independent and truly uniform hash functions. For $x \in \mathcal{U}$, define the linear sequence $f_1(x) = f(x)$, and $f_{i+1}(x) = f_i(x) + 1 \mod n$, for $i \in [n]$; and similarly define the sequence $g_i(x)$. The algorithm, then, inserts each key x into the first unoccupied cell in the following probe sequence: $f_1(x)$, $g_1(x)$, $f_2(x)$, $g_2(x)$, $f_3(x)$, $g_3(x)$, ... We denote this algorithm that hashes m keys into n cells by ShortSeq(n, m), for the shortest sequence.

The second algorithm inserts each key into the empty (terminal) cell that is the right neighbor of the smallest cluster among the two clusters containing the initial hashing cells, breaking ties randomly. If one of the initial cells is empty, then the key is inserted into it, and if both of the initial cells are empty, we break ties evenly. Recall that a cluster is a group of consecutively occupied cells whose left and right neighbors are empty cells. This means that one can compute the size of the cluster that contains an initial hashing cell by running two linear probe sequences in opposite directions

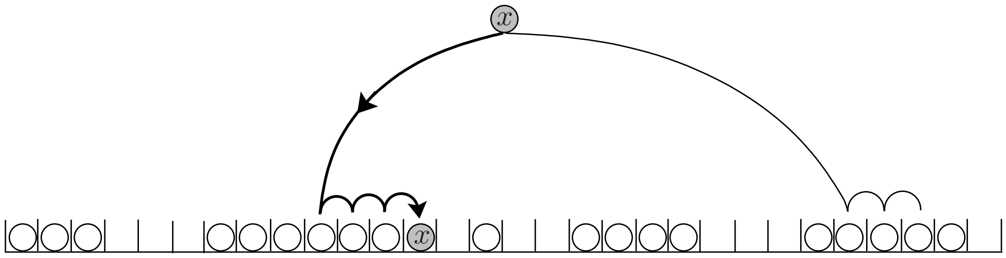

Figure 5.1: An illustration of algorithm ShortSeq(n, m) in terms of balls (keys) and bins (cells). Each ball is inserted into the empty bin founded by the shortest sequence.

starting from the initial cell and going to the empty cells at the boundaries. So practically, the algorithm uses four linear probe sequences. We refer to this algorithm by SmallCluster(n, m) for inserting m keys into n cells.

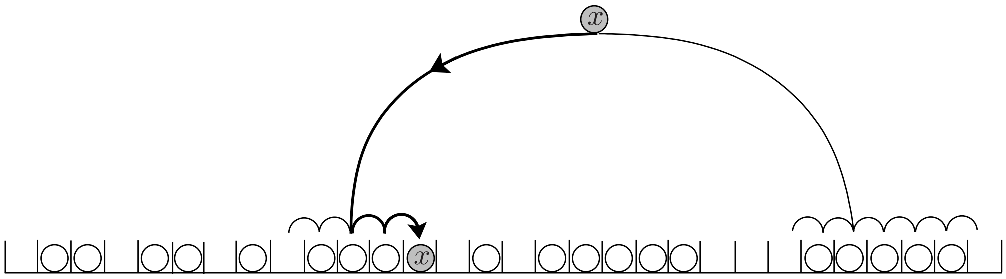

Figure 5.2: Algorithm SmallCluster(n, m) inserts each key into the empty cell adjacent to the smallest cluster, breaking ties randomly. The size of the clusters is determined by probing linearly in both directions.

The performances of algorithms ShortSeq and SmallCluster, as we are going to see, is unexpectedly disappointing. The main mistake in these two algorithms is that the keys are allowed to be inserted into empty cells even if these cells are very close to some big clusters.

Theorem 5.4. *Let $\epsilon \in (0,1)$ be an arbitrary constant, and $n, m \in \mathbb{N}$ be such that $\alpha := m/n \in (0,1)$ is a constant. Let $\mathbf{A}$ be a two-way linear probing algorithm that inserts m keys into n cells such that whenever a key chooses an empty and an occupied initial cells, the algorithm inserts the key into the empty one. Then algorithm $\mathbf{A}$ produces a cluster of size at least $(1-\epsilon)\log n/\log\log n$, w.h.p.*

Proof. Set $\xi(n) := \lceil (1-\epsilon)\log n/\log\log n \rceil$. Let $\beta = \lfloor \log n \rfloor$, and assume, without loss of generality, that $N := n/\beta$ is an integer. We say "at time t" to mean immediately after the insertion of t keys. For $t \in [m]$, and $i \in [N]$, let $X_i(t)$ be the number of consecutively occupied cells at time t that occur between the cell number βi and the first empty cell that comes after it. More precisely, $X_i(t) = j$ if and only if at time t, all the cells $\beta i + 1, \ldots, \beta i + j$ are occupied, and the cell $\beta i + j + 1$ is empty, where we consider the cell numbers in a circular fashion. Let $\chi_i(t) := \beta i + X_i(t) + 1$, and notice that the cell $\chi_i(t)$ is always empty, because $m < n$. Clearly, if all clusters are smaller than ξ, then $\max_i X_i(m) < \xi$. So we only need to show that $\mathbb{P}\{\max_i X_i(m) < \xi\} = o(1)$. Setting $X_i(0) = 0$, we can write

$$X_i(m) = \sum_{t=1}^{m} \mathbb{I}_{[\, X_i(t) = X_i(t-1)+1 \,]} \,.$$

For $i \in [N]$, and $t \in [m]$, let $Y_i(t)$ be the indicator that the first initial cell of the t-th key is the empty cell $\chi_i(t-1)$, and the second initial cell is an occupied cell. By assumption, if $Y_i(t) = 1$, then $X_i(t) = X_i(t-1) + 1$, because the algorithm inserts the t-th key into the cell $\chi_i(t-1)$. Therefore, for all $i \in [N]$, we have

$$X_i(m) \geq \sum_{t=\lceil m/2 \rceil}^{m} Y_i(t) \stackrel{\text{def}}{=} Z_i \,.$$

Notice that the random variables $Z_1, \ldots, Z_N$ are negatively associated when conditioned on the event $A := [\, \max_i X_i(m) < \beta \,]$, which can been seen as follows. For $t \in [m]$, let $Y_0(t) := 1 - \sum_{i=1}^{N} Y_i(t)$, and notice that given A, the random variable

$Y_0(t)$ is binary. Lemma 0.4 says that the binary random variables $Y_0(t), \ldots, Y_N(t)$ are negatively associated when conditioned on A. However, since the keys choose their initial cells independently, the random variables $Y_0(t), \ldots, Y_N(t)$ are mutually independent from the random variables $Y_0(t'), \ldots, Y_N(t')$, for any distinct $t, t' \in [m]$. Thus, by Lemma 0.5, the union $\cup_{t=1}^m \{Y_0(t), \ldots, Y_N(t)\}$ is a set of negatively associated random variables under the same conditioning. Now the conditional negative association of the Z_i is assured by Lemma 0.6. Consequently, we have

$$\begin{aligned}
\mathbb{P}\left\{\max_i X_i(m) < \xi\right\} &= \mathbb{P}\left\{\left[\max_i X_i(m) < \xi\right] \cap A\right\} \leq \mathbb{P}\left\{\max_i Z_i < \xi \,\middle|\, A\right\} \\
&\leq \mathbb{P}\{Z_1 < \xi, \ldots, Z_N < \xi \mid A\} \\
&\leq \prod_{i=1}^N (1 - \mathbb{P}\{Z_i \geq \xi \mid A\}) \\
&\leq \exp\left(-\sum_{i=1}^N \mathbb{P}\{Z_i \geq \xi \mid A\}\right),
\end{aligned}$$

which goes to zero if $\sum_{i=1}^N \mathbb{P}\{Z_i \geq \xi \mid A\} \xrightarrow{n} \infty$. Since the keys choose their initial cells independently and uniformly at random, we see that for all $t \geq \lceil m/2 \rceil$, and n large enough,

$$\mathbb{P}\{Y_i(t) = 1 \mid A\} \geq \frac{t-1}{n^2} \geq \frac{\alpha}{4n}.$$

Thus, by the independence of $Y_i(1), \ldots, Y_i(m)$, for each $i \in [N]$, and the binomial tail inequality of Lemma 0.2, we see that

$$\sum_{i=1}^N \mathbb{P}\{Z_i \geq \xi \mid A\} \geq \sum_{i=1}^N \mathbb{P}\{\mathrm{Bin}(\lceil m/2 \rceil, \alpha/(4n)) \geq \xi\} \geq cN\left(\frac{\alpha^2}{16\,\xi}\right)^{\xi} = \omega(1),$$

for some constant $c > 0$. □

Clearly, algorithms ShortSeq(n, m) and SmallCluster(n, m) satisfy the condition of Theorem 5.4. So this corollary follows.

Corollary 5.1. *Let $n, m \in \mathbb{N}$, and assume that $\alpha := m/n \in (0, 1)$ is a constant. The size of the largest cluster generated by algorithm* ShortSeq(n, m) *is at least*

$(1-\epsilon)\log n/\log\log n$, *w.h.p., for any constant* $\epsilon \in (0,1)$. *The same result holds for algorithm* SMALLCLUSTER(n, m).

It is worth mentioning that simulation results of algorithms SHORTSEQ and SMALLCLUSTER shows that the worst-case performance of SMALLCLUSTER is better than SHORTSEQ. This is somehow expected as the algorithm considers more information before it makes its decision of where to insert the keys.

Chapter 6

New Paradigms for Two-way Linear Probing

We propose new efficient two-way linear probing algorithms with remarkable worst-case performances. The common idea of these algorithms is the marriage between the concept of two-way linear probing and a technique we call *blocking* where the hash table is partitioned into equal-sized blocks of cells. Whenever a key has two terminal cells, the algorithm considers the information provided by the blocks, e.g., the number of keys it harbors, to make a decision. Thus, the blocking technique enables the algorithm to avoid some of the bad decisions the previous algorithms, described in the last chapter, make. This leads to a more controlled allocation process, and hence, to a more even distribution of the keys. We use the blocking technique to design, in Sections 6.2 and 6.3, two two-way linear probing algorithms. In Section 6.1, we give a simple algorithm that uses linear probing locally within each block. The algorithms, which are implemented with FCFS replacement strategy, are characterized by the way the keys pick their blocks to land in.

The maximum unsuccessful search times of these algorithms are analyzed and proven to be $O(\log\log n)$, asymptotically almost surely, which can be viewed in conjunction with the universal lower bound we proved in Section 5.3. Simulation results, provided in Section 6.5, supports our theoretical analyses of all algorithms discussed here, in addition to the ones in Chapter 5. Furthermore, the memory space consumption is still linear. Although we assume throughout that these algorithms keep a counter with each block, the extra space consumed by these counters is asymptotically sublinear. In fact, we will see that the extra space is $O(n/\log\log n)$ in a model in which integers take $O(1)$ space, and at worst $O(n\log\log\log n/\log\log n) = o(n)$, w.h.p., in a bit model.

Since the block size for each of the following algorithms is different, we assume throughout and without loss of generality, that whenever we use a block of size β, then n/β is an integer. Recall that the hash table $\mathcal{T} = \{0, \ldots, n-1\}$, and hence, for $i \in [n/\beta]$, the i-th block consists of the cells $(i-1)\beta, \ldots, i\beta - 1$. In other words, the cell $k \in \mathcal{T}$ belongs to block number $\lambda(k) := \lfloor k/\beta \rfloor + 1$.

6.1 Two-way Locally-linear Probing

As a simple example of the blocking technique, we present the following algorithm which is a trivial application of the two-way chaining scheme we study in Chapters 1 and 2. The algorithm does not satisfy the conditions of two-way linear probing as explained in Definition 5.1, because the linear probes are performed within each block and not along the hash table; so they are locally linear within the blocks. That is, whenever the linear probe sequence reaches the right boundary of a block, it continues probing starting from the left boundary of the same block.

The algorithm is described as follows. We partition the hash table into disjoint blocks each of size $\beta_1(n)$, where $\beta_1(n)$ is an integer to be defined later. The *load of a block* is defined to be the number of keys residing in its cells. We save with each block its load, and keep it updated whenever a key is inserted in the block. For each key we choose two initial hashing cells, and hence two blocks, independently and uniformly at random, with replacement. From the initial cell that belongs to the least loaded block, breaking ties randomly, we probe linearly and cyclically *within the block* until we find an empty cell where we insert the key. If the load of the block is β_1, i.e., it is full, then we check its right neighbor block and so on, until we find a block that is not completely full. We insert the key into the first empty cell there. Notice that only one probe sequence is used to insert any key. The search operation, however, uses two probe sequences as follows. First, we compute the two initial hashing cells. We start probing linearly, cyclically and alternately within the two (possibly identical) blocks that contain these initial cells. If both probe sequences reach empty cells, or if one of them reaches an empty cell and the other one finishes the block without finding the key, we declare that it is unsuccessful search. If both blocks are full and the probe sequences completely search them without finding the key, then the right neighbors of these blocks (cyclically speaking) are searched sequentially, and so on, until the key is found or two empty cells in the case of unsuccessful search. We will refer to this algorithm as LocallyLinear(n, m), where there are m keys and n cells. We show next that β_1 can be defined such that none of the blocks are completely full, w.h.p. This means that whenever we search for any key, most of the time, we only need to search linearly and cyclically the two blocks the key chooses initially.

Theorem 6.1. *Let $n, m \in \mathbb{N}$, where $\alpha = m/n \in (0, 1)$ is a constant. Let C be the constant defined in Theorem 0.3, and define*

$$\beta_1(n) := \left\lfloor \frac{\log_2 \log n + C}{1 - \alpha} + 1 \right\rfloor .$$

Then, w.h.p., the maximum unsuccessful search time of LocallyLinear(n, m) *with blocks of size β_1 is at most $2\beta_1$, and the maximum insertion time is at most $\beta_1 - 1$. The bounds are tight up to additive constants.*

Proof. Notice the equivalence between algorithm LocallyLinear(n, m) and the allocation process Uniform-GreedyMC$(n/\beta_1, m)$ where m balls (keys) are inserted into n/β_1 bins (blocks) by placing each ball into the least loaded bin among two bins chosen independently and uniformly at random, with replacement, where ties are broken randomly. It suffices, therefore, to study the maximum bin load of Uniform-GreedyMC$(n/\beta_1, m)$ which we denote by L_n. However, Theorem 0.3 says that w.h.p.,

$$L_n \leq \log_2 \log n + C + \alpha\beta_1 < (1-\alpha)\beta_1 + \alpha\beta_1 = \beta_1 .$$

and similarly,

$$L_n \geq \log_2 \log n + \alpha\beta_1 - C > \frac{\log_2 \log n + C}{1-\alpha} - 2C \geq \beta_1 - 2C - 1 .$$

□

6.2 Two-way Pre-linear Probing

In the two-way linear probing algorithms of Chapter 5, each input key initiates two linear probe sequences that reach two terminal cells, and then the algorithms decide in which terminal cell the key should be inserted. The following algorithm, however, allows each key to choose two initial hashing cells, and then decides, according to some strategy, which initial cell should start a linear probe sequence to find a terminal cell to harbor the key. So, technically, the insertion process of any key uses only one linear probe sequence, but we still use two sequences for any search. The following algorithm is similar to algorithm LocallyLinear.

Let $\alpha \in (0, 1)$ be the load factor. Partition the hash table into blocks of size $\beta_2(n)$, where $\beta_2(n)$ is an integer to be defined later. Each key x still chooses, independently and uniformly at random, two initial hashing cells, say I_x and J_x, and hence, two blocks which we denote by $\lambda(I_x)$ and $\lambda(J_x)$. For convenience, we say that the key

x has *landed* in block i, if the linear probe sequence used to insert the key x has started (from the initial hashing cell available for x) in block i. Define the *weight of a block* to be the number of keys that have landed in it. We save with each block its weight, and keep it updated whenever a key lands in it. Now, upon the arrival of key x, the algorithm allows x to land into the block, among $\lambda(I_x)$ and $\lambda(J_x)$, of smallest weight, breaking ties randomly. Whence, it starts probing linearly from the initial cell contained in the block until it finds a terminal cell into which the key x is placed. If, for example, both I_x and J_x belong to the same block, then x lands in $\lambda(I_x)$, and the linear sequence starts from an arbitrarily chosen cell among I_x and J_x. We will write DECIDEFIRST(n, m) to refer to this algorithm for inserting m keys into n cells.

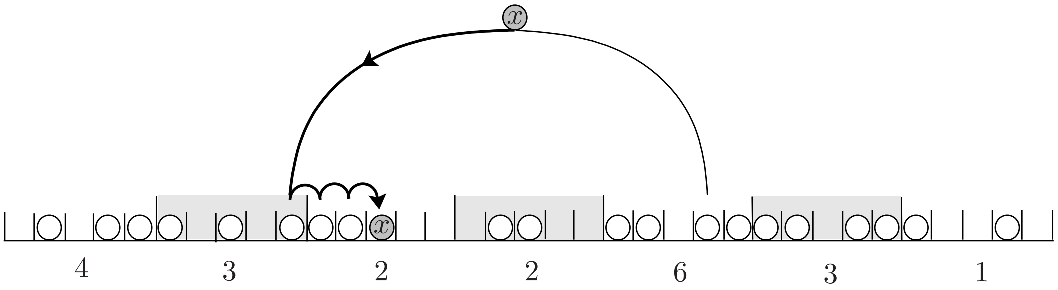

Figure 6.1: An illustration of algorithm DECIDEFIRST(n, m). The hash table is divided into blocks of size β_2. The number under each block is its weight. Each key decides first to land into the block of smallest weight, breaking ties randomly, then probes linearly to find its terminal cell.

In short, the strategy of DECIDEFIRST(n, m) is: land in the block of smallest weight, walk linearly, and insert into the first empty cell reached. The size of the largest cluster produced by the algorithm is $\Theta(\log\log n)$. The performance of this hashing technique is described in the following theorem.

Theorem 6.2. *Let $\alpha = m/n \in (0, 1)$ be constant, where $n, m \in \mathbb{N}$. There is a* *constant* $\eta > 0$ *such that if*

$$\beta_2(n) := \left\lceil \frac{(1+\sqrt{2-\alpha})}{\sqrt{2-\alpha}(1-\alpha)}(\log_2 \log n + \eta) \right\rceil ,$$

then, w.h.p., the maximum unsuccessful search time of algorithm DecideFirst(n, m) *with blocks of size* β_2 *is at most* $\xi_n := 12(1-\alpha)^{-2}(\log_2 \log n + \eta)$, *and the maximum insertion time is at most* $\xi_n/2$.

Proof. Assume first that DecideFirst(n, m) is applied to a hash table with blocks of size $\beta = \lceil b(\log_2 \log n + \eta) \rceil$, and that n/β is an integer, where $b = (1+\epsilon)/(1-\alpha)$, for some arbitrary constant $\epsilon > 0$. Consider the resulting hash table after termination of the algorithm. Let $M \geq 0$ be the maximum number of consecutive blocks that are fully occupied. Without loss of generality, suppose that these blocks start at block $i+1$, and let $S = \{i, \ldots, i+M\}$ represent these full blocks in addition to the left adjacent block that is not fully occupied (Figure 6.2).

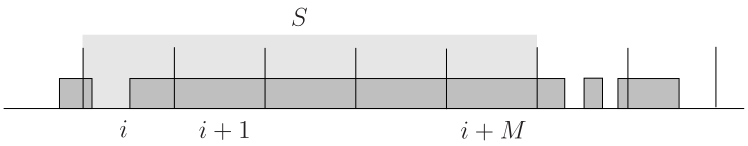

Figure 6.2: A portion of the hash table showing the largest cluster, and the set S which consists of the full consecutive blocks and their left neighbor.

Notice that each key chooses two cells (and hence, two possibly identical blocks) independently and uniformly at random. Also, any key always lands in the block of smallest weight. Since there are n/β blocks, and $m = \alpha n$ keys, then by Theorem 0.3, there is a constant $C > 0$ such that the maximum block weight is not more than $\lambda_n := (\alpha b + 1)\log_2 \log n + \alpha b \eta + \alpha + C$, w.h.p. Let A_n denote the event that the maximum block weight is at most λ_n. Let W be the number of keys that have landed in S, i.e., the total weight of blocks contained in S. Plainly, since block i is not full, then all the keys that belong to the M full blocks have landed in S. Thus, $W \geq Mb(\log_2 \log n + \eta)$, deterministically. If we choose $\eta = C + \alpha$, then the event A_n implies that $(M+1)(\alpha b + 1) \geq Mb$, because otherwise, we have

$$W \leq (M+1)(\alpha b+1)\left(\log_2 \log n + \frac{\alpha b \eta + \alpha + C}{\alpha b + 1}\right) < Mb(\log_2 \log n + \eta)\,,$$

which is a contradiction. Therefore, A_n yields that

$$M \leq \frac{\alpha b + 1}{(1-\alpha)b - 1} \leq \frac{1 + \epsilon\alpha}{\epsilon(1-\alpha)}\,.$$

Recall that $(\alpha b + 1) < b = (1+\epsilon)/(1-\alpha)$. Again, since block i is not full, the size of the largest cluster is not more than the total weight of the $M+2$ blocks that cover it. Consequently, the maximum cluster size is, w.h.p., not more than

$$(M+2)(\alpha b + 1)(\log_2 \log n + \eta) \leq \frac{\psi(\epsilon)}{(1-\alpha)^2}(\log_2 \log n + \eta)\,,$$

where $\psi(\epsilon) := 3 - \alpha + (2-\alpha)\epsilon + 1/\epsilon$. Since ϵ is arbitrary, we choose it such that $\psi(\epsilon)$ is minimum, i.e., $\epsilon = 1/\sqrt{2-\alpha}$; in other words, $\psi(\epsilon) = 3 - \alpha + 2\sqrt{2-\alpha} < 6$. Since the maximum unsuccessful search time is at most twice the maximum cluster size plus two, the result follows for n large enough. □

Remark 6.1. We have shown that w.h.p. the maximum cluster size produced by DecideFirst(n, m) is in fact not more than

$$\frac{3 - \alpha + 2\sqrt{2-\alpha}}{(1-\alpha)^2}\log_2 \log n + O(1) < \frac{6}{(1-\alpha)^2}\log_2 \log n + O(1)\,.$$

6.3 Two-way Post-linear Probing

We introduce yet another hashing algorithm that achieves $\Theta(\log \log n)$ worst-case search time, a.a.s. Suppose that the hash table is divided into blocks of size $\beta_3(n)$, where $\beta_3(n)$ is an integer to be picked later. Recall that the load of a block is the number of keys (or occupied cells) it contains. Suppose that we save with each block its load, and keep it updated whenever a key is inserted into one of its cells. Now the algorithm works as follows. Each key x gets two initial hashing cells. From these initial cells the algorithm probes linearly and cyclically until it finds two empty cells U_x and V_x, which we call terminal cells. Let $\lambda(U_x)$ and $\lambda(V_x)$ be the blocks that contain these cells. The algorithm, then, inserts the key x into the terminal cell (among U_x and V_x) that belongs to the least loaded block among $\lambda(U_x)$ and $\lambda(V_x)$, breaking ties randomly. We refer to this algorithm for inserting m keys into n cells as WalkFirst(n, m). Notice that since the algorithm uses both linear sequences to insert any key, the construction time here is, roughly speaking, twice the one in DecideFirst(n, m).

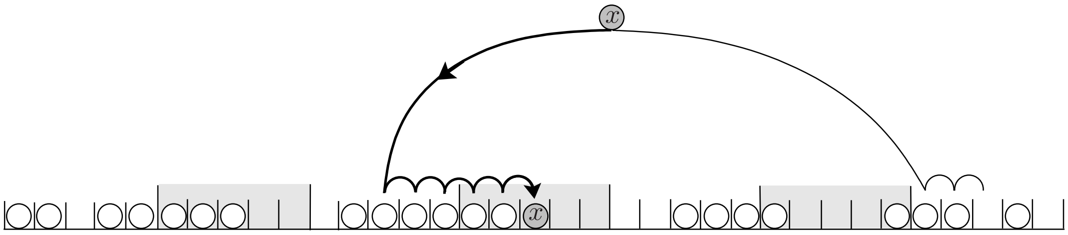

Figure 6.3: Algorithm WalkFirst(n, m) inserts each key into the terminal cell the belongs to the least crowded block, breaking ties arbitrarily.

In the remainder of this section, we analyze the worst-case performance of algorithm WalkFirst(n, m). Recall that the maximum unsuccessful search time is bounded from above by twice the maximum cluster size plus two. The following theorem asserts that upon termination of the algorithm, it is most likely that every block has at least one empty cell. This implies that the length of the largest cluster is at most $2\beta_3 - 2$.

Theorem 6.3. *Let $n, m \in \mathbb{N}$ such that $\alpha = m/n \in (0, 1/2)$ is a constant. Let*

$\delta \in (2\alpha, 1)$ be an arbitrary constant, and define

$$\beta_3(n) := \left\lceil \frac{\log_2 \log n + 8}{1 - \delta} \right\rceil .$$

Upon termination of algorithm WalkFirst(n, m) *with blocks of size β_3, the probability that there is a fully loaded block goes to zero as n tends to infinity. That is, w.h.p., the maximum unsuccessful search time of* WalkFirst(n, m) *is at most $4\beta_3 - 2$, and the maximum insertion time is at most $4\beta_3 - 4$.*

For $k \in [m]$, let us denote by A_k the event that after the insertion of k keys (i.e., at time k), none of the blocks is fully loaded. To prove Theorem 6.3, we need to show that $\mathbb{P}\{A_m^c\} = o(1)$. We do that by using a witness tree argument just like the one we used to prove Theorems 1.4 and 2.5. We show that if a fully-loaded block exists, then there is a witness binary tree of height β_3 that describes the history of that block. But first, we explain how to construct a witness tree. Detailed description of the witness tree, and illustrated figures can be found in Section 1.4. The witness tree has been used also in [170, 134, 33, 34, 127].

Recall that we have m keys which are inserted into n/β_3 blocks. The witness tree we defined in Section 1.4 is in terms of balls and bins, but one can easily translate it to keys and blocks. Observe that the definition, as we are going to see next, is independent of the number of bins or blocks. Let us number the keys $1, \ldots, m$ according to their insertion time. Recall that each key $t \in [m]$ has two initial cells which lead to two terminal empty cells belonging to two blocks. Let us denote these two blocks available for the t-th key by X_t and Y_t. Notice that all the initial cells are independent and uniformly distributed. However, all terminal cells—and so their blocks—are not. Nonetheless, for each fixed t, the two random values X_t and Y_t are independent.

The History Tree

We define for each key t a *full history tree* T_t in the same way as we did in the balls-and-bins model. Each key is identified with the block that contains it, and the full history tree T_t describes the history of the block that contains the t-th key up to its insertion time. The root of T_t is labelled t, and is colored white. The root has two children, a left child corresponding to the block X_t, and a right child corresponding to the block Y_t. The left child is labelled and colored according to the following rules:

(a) If the block X_t contains some keys at the time of insertion of key t, and the last key inserted in that block, say τ, has not been encountered thus far in the (BFS) order of T_t, then the node is labelled τ and colored white.

(b) As in case (a), except that τ has already been encountered in the BFS order. The node is labelled τ, but colored black.

(c) If the block X_t is empty at the time of insertion of key t, then the left child is unlabelled gray node.

Similarly, the right child of t is labelled and colored by following the same rules but with the block Y_t. We continue recursively constructing the tree until all the leaves are black or gray. A black or gray node in the tree is a leaf and is not processed any further. A white node with label τ is processed in the same way we processed the key t, but with its two blocks X_τ and Y_τ.

Since we are using the same definition of the full history tree, all the properties we mentioned in Section 1.4 are also valid here. In particular, the tree T_t has at least one gray leaf, every internal (white) node has two children, and the length of the shortest path from the root t to any gray node is equal to the load of the block that contains the key t at the time of its insertion. Thus, if the block's load is more than h, then all gray nodes must be at distance more than h from the root. Subsequently, we define the *truncated history tree* of height h, that is, with $h+1$ levels of nodes, to be the top part of the full history tree that includes all nodes at the first $h+1$ levels only, and the remainder is truncated. Most importantly, we are interested in truncated history trees of height h where the nodes at the lowest level are either black nodes, or white nodes that represent blocks of loads at least $\xi > 0$, where ξ is an integer to be chosen later. We call every white node at the lowest level a "block" node. Evidently, these special truncated history trees describe blocks of load at least $h+\xi$.

The Witness Tree

Similarly, we define the witness tree. Let $\xi \in \mathbb{N}$ be a fixed integer to be decided later. For $h, k \in \mathbb{N}$, where $h+\xi \leq k \leq m$, a *witness tree* $W_k(h)$ is a truncated history tree of a key in the set $[k]$, with $h+1$ levels of nodes (thus, of height h) and with two types of leaf nodes, black nodes and unlabelled block nodes which represent blocks with load of at least ξ. The node labels belong to the set $[k]$. Each black leaf has a label of an internal (white) node that precedes it in BFS. Block nodes must all be at the furthest level from the root, and there is at least one such node in a witness tree. For any $k, h, d \in \mathbb{N}$, and nonnegative integer z, we write $\mathcal{W}_k(h, d, z)$ to denote the class of all witness trees $W_k(h)$ of height h that have d white nodes, and z black nodes (and thus $d-z+1$ block nodes). Notice that $d \in [h, 2^h)$ and $z \in [0, d]$. We say that a witness tree $W_k(h)$ *occurs*, if upon execution of algorithm WalkFirst, the random choices available for the keys represented by the witness tree are actually as indicated in the witness tree itself. Thus, a witness tree of height h exists if and only if there is a key that is inserted by algorithm WalkFirst(n, m) into a block whose load is at least $h+\xi-1$, just before the insertion of the key.

Thus far, we have only translated the original definition of the witness tree to a one that deals with keys and blocks. Therefore, we can safely use here Lemmas 1.1

and 1.2. We only need now to bound the probability that a valid witness tree occurs.

Lemma 6.1. *Let D denote the event that the number of blocks in* WalkFirst(n,m) *with load of at least ξ, after termination, is at most $n/(a\beta_3\xi)$, for some constant $a>0$. For $k\in[m]$, let A_k be the event that after the insertion of k keys, none of the blocks is fully loaded. Then for any positive integers h,d and $k\geq h+\xi$, and a non-negative integer $z\leq d$, we have*

$$\sup_{W_k(h)\in\mathcal{W}_k(h,d,z)}\mathbb{P}\left\{W_k(h)\ occurs \mid A_{k-1}\cap D\right\}\leq\frac{4^d\beta_3^{d+z-1}}{(a\xi)^{d-z+1}n^{d+z-1}}\,.$$

Proof. Let X_t and Y_t be the blocks available for key $t\in[k]$. Recall that we have $\nu:=n/\beta_3$ blocks. Given that we know the history up to time $t-1$, and the events A_{k-1} and D are true, the probability that X_t is the i-th block is at most $2/\nu$, because the initial hashing cell (which is chosen independently and uniformly at random) has to be in the $(i-1)$-th or the i-th blocks; and hence, the probability that X_t is a block of load at least ξ is at most $2/(a\xi)$, as there are at most $\nu/(a\xi)$ such blocks. The result now follows by the conditional probability method, just like as we did in the proof of Lemma 2.1. □

The next step is to show that the event D in Lemma 6.1 is most likely to be true, for sufficiently large $\xi<\beta_3$. Define the function $\varphi(x)=x^{-1}e^{1-1/x}$, for $x>1$. Notice that φ is decreasing on $(1,\infty)$, and for any $x>1$, we have $\varphi(x)<1$, because $1/x=(1-z)<e^{-z}=e^{1/x-1}$, for some $z\in(0,1)$. Thus, by inequality (2) of Lemma 0.1, we see that for $p\in(0,1)$, and any positive integers r, and $t\geq\eta rp$, for some $\eta>1$, we have

$$\mathbb{P}\left\{\mathrm{Bin}(r,p)\geq t\right\}\leq\left(\varphi\left(\frac{t}{rp}\right)\right)^t\leq(\varphi(\eta))^t\,. \tag{6.1}$$

Lemma 6.2. *Let α, δ, and β_3 be as defined in Theorem 6.3. Let N be the number of blocks with load of at least ξ upon termination of algorithm* WalkFirst(n,m). *If $\xi\geq\delta\beta_3$, then $\mathbb{P}\left\{N\geq n/(a\beta_3\xi)\right\}=o(1)$, for any constant $a>0$.*

Proof. Fix $\xi \geq \delta\beta_3$. Let B denote the last block in the hash table, i.e., B consists of the cells $n-\beta_3, \ldots, n-1$. Let L be the load of B after termination. Since the loads of the blocks are identically distributed, we have

$$\mathbf{E}[\,N\,] = \frac{n}{\beta_3}\mathbb{P}\{L \geq \xi\}\ .$$

Let S be the set of the consecutively occupied cells, after termination, that occur between the first empty cell to the left of the block B and the cell $n-\beta_3$.

Figure 6.4: The last part of the hash table showing clusters, the last block B, and the set S.

We say that a key is *born* in a set of cells A if at least one of its two initial hashing cells belong to A. For convenient, we write $\Phi(A)$ to denote the number of keys that are born in A. Obviously, $\Phi(A) \stackrel{\mathcal{L}}{=} \text{Bin}(m, 2\,|A|\,/n)$. Since the cell adjacent to the left boundary of S is empty, all the keys that are inserted in S are actually born in S. That is, if $|S| = i$, then $\Phi(S) \geq i$. So, by inequality (6.1), we see that

$$\mathbb{P}\{|S| = i\} = \mathbb{P}\{[\,\Phi(S) \geq i\,] \cap [\,|S| = i\,]\} \leq \mathbb{P}\{\text{Bin}(m, 2i/n) \geq i\} \leq c^i\,, \qquad (6.2)$$

where the constant $c := \varphi(1/(2\alpha)) = 2\alpha e^{1-2\alpha} < 1$, because $\alpha < 1/2$. Let

$$k := \log_c \frac{1-c}{\xi^2} = O(\log \beta_3)\,.$$

and notice that for n large enough,

$$\xi \geq \delta\beta_3 = \frac{\delta 2m(k+\beta_3)}{(1+k/\beta_3)2\alpha n} \geq y\frac{2m(k+\beta_3)}{n}\,,$$

where $y = 1/2 + \delta/(4\alpha) > 1$, because $\delta \in (2\alpha, 1)$. Clearly, by the same property of S stated above, $L \leq \Phi(S \cup B)$; and hence, using inequality (6.1) again, we conclude that for n sufficiently large,

$$\begin{aligned}
\mathbb{P}\{L \geq \xi\} &\leq \mathbb{P}\{[\,\Phi(S \cup B) \geq \xi\,] \cap [\,|S| \leq k\,]\} + \sum_{i=k}^{m} \mathbb{P}\{|S| = i\} \\
&\leq \mathbb{P}\{\mathrm{Bin}(m, 2(k+\beta_3)/n) \geq \xi\} + \frac{c^k}{1-c} \\
&\leq (\varphi(y))^{\xi} + \frac{c^k}{1-c} \;\leq\; \frac{1}{\xi^2} + \frac{1}{\xi^2} \;=\; \frac{2}{\xi^2}\,.
\end{aligned}$$

Thence, $\mathbf{E}\,[\,N\,] \leq 2n/(\beta_3\xi^2)$ which implies by Markov's inequality that

$$\mathbb{P}\left\{N \geq \frac{n}{a\beta_3\xi}\right\} \leq \frac{2a}{\xi} = o(1)\,.$$

□

Proof of Theorem 6.3.

Recall that A_k, for $k \in [m]$, is the event that after the insertion of k keys (i.e., at time k), none of the blocks is fully loaded. Notice that $A_m \subseteq A_{m-1} \subseteq \cdots \subseteq A_1$, and the event A_{β_3-1} is deterministically true. We shall show that $\mathbb{P}\{A_m^c\} = o(1)$. Let D denote the event that the number of blocks with load of at least ξ, after termination, is at most $n/(a\beta_3\xi)$, for some constant $a > 1$ to be decided later. Observe that

$$\begin{aligned}
\mathbb{P}\{A_m^c\} &\leq \mathbb{P}\{D^c\} + \mathbb{P}\{A_m^c \,|\, D\} \\
&\leq \mathbb{P}\{D^c\} + \mathbb{P}\{A_m^c \,|\, A_{m-1} \cap D\} + \mathbb{P}\left\{A_{m-1}^c \,|\, D\right\} \\
&\;\;\vdots \\
&\leq \mathbb{P}\{D^c\} + \sum_{k=\beta_3}^{m} \mathbb{P}\{A_k^c \,|\, A_{k-1} \cap D\}\,.
\end{aligned}$$

Lemma 6.2 reveals that $\mathbb{P}\{D^c\} = o(1)$, and hence, we only need to demonstrate that $p_k := \mathbb{P}\{A_k^c \,|\, A_{k-1} \cap D\} = o(1/n)$, for $k = \beta_3, \ldots, m$. We do that by using the witness tree argument. The proof is basically similar to the one of Theorem 2.5. Let $h, \xi, \eta \in [2, \infty)$ be some integers to be picked later such that $h + \xi \leq \beta_3$. If after the insertion of k keys, there is a block with load of at least $h+\xi$, then a witness tree $W_k(h)$ (with block nodes representing blocks with load of at least ξ) must have occurred. Recall that $d \in [2, 2^h)$ and $z \in [0, d]$. Using Lemmas 1.1, 1.2, and 6.1, we see that

$$\begin{aligned}
p_k &\leq \sum_{d=2}^{2^h-1} \sum_{z=0}^{d} \sum_{W_k(h)\in\mathcal{W}_k(h,d,z)} \mathbb{P}\left\{W_k(h) \text{ occurs} \,|\, A_{k-1}\cap D\right\} \\
&\leq \sum_{d=2}^{2^h-1} \sum_{z=0}^{d} |\mathcal{W}_k(h,d,z)| \sup_{W_k(h)\in\mathcal{W}_k(h,d,z)} \mathbb{P}\left\{W_k(h) \text{ occurs} \,|\, A_{k-1}\cap D\right\} \\
&\leq \sum_{d=2}^{2^h} \sum_{z=0}^{d} \frac{2^{d+1}4^{2d}d^z k^d \beta_3^{d+z-1}}{(a\xi)^{d-z+1}\, n^{d+z-1}} \mathbb{I}_{\left[\,[z\geq\eta]\cup[d>2^{h-\eta}]\,\right]} \\
&\leq \frac{2n}{a\xi\beta_3} \sum_{d=2}^{2^h} \left(\frac{32\alpha\beta_3}{a\xi}\right)^d \sum_{z=0}^{d} \left(\frac{ad\xi\beta_3}{n}\right)^z \mathbb{I}_{\left[\,[z\geq\eta]\cup[d>2^{h-\eta}]\,\right]}\,.
\end{aligned}$$

We split the sum over $d \leq 2^{h-\eta}$, and $d > 2^{h-\eta}$. For $d \leq 2^{h-\eta}$, we have $z \geq \eta$, and thus

$$\begin{aligned}
\sum_{z=0}^{d} \left(\frac{ad\xi\beta_3}{n}\right)^z \mathbb{I}_{\left[\,[z\geq\eta]\cup[d>2^{h-\eta}]\,\right]} &= \sum_{z=\eta}^{d} \left(\frac{ad\xi\beta_3}{n}\right)^z \\
&\leq \left(\frac{ad\xi\beta_3}{n}\right)^\eta \sum_{z=0}^{\infty} \left(\frac{ad\xi\beta_3}{n}\right)^z \\
&< 2\left(\frac{ad\xi\beta_3}{n}\right)^\eta,
\end{aligned}$$

provided that n is so large that $a2^{h+1}\xi\beta_3 \leq n$, (this insures that $ad\xi\beta_3/n < 1/2$). For $d \in (2^{h-\eta}, 2^h]$, we bound trivially, assuming the same large n condition:

$$\sum_{z=0}^{d} \left(\frac{ad\xi\beta_3}{n}\right)^z \leq 2\,.$$

In summary, we see that

$$p_k \leq 4n \sum_{d>2^{h-\eta}} \left(\frac{32\alpha\beta_3}{a\xi}\right)^d + 4\left(\frac{a\xi\beta_3}{n}\right)^{\eta-1} \sum_{d=2}^{2^{h-\eta}} \left(\frac{32\alpha\beta_3}{a\xi}\right)^d d^\eta\,.$$

We set $a = 32$, and $\xi = \lceil \delta\beta_3 \rceil$, so that $32\alpha\beta_3/(a\xi) \le 1/2$, because $\delta \in (2\alpha, 1)$. With this choice, we have

$$p_k \le \frac{4n}{2^{2^{h-\eta}}} + 4c\left(\frac{32\beta_3^2}{n}\right)^{\eta-1},$$

where $c = \sum_{d\ge 2} d^\eta/2^d$. Clearly, if we put $h = \eta + \lceil \log_2 \log_2 n^\eta \rceil$, and $\eta = 3$, then we see that $h + \xi \le \beta_3$, and $p_k = o(1/n)$. Notice that h and ξ satisfy the technical condition $a2^{h+1}\xi\beta_3 \le n$, asymptotically. $\square$

Remark 6.2. The restriction on α is needed only to prove Lemma 6.2 where the binomial tail inequality is valid only when $\alpha < 1/2$. Simulation results suggest that Theorem 6.3 is, indeed, true for any $\alpha \in (0, 1)$ with block size $\lfloor (1-\alpha)^{-1}(\log_2 \log n + c) \rfloor$, for some constant c.

6.4 Other Variants

Speedups and Trade-offs

We have seen that by using two linear probe sequences instead of just one, the maximum unsuccessful search time decreases exponentially from $O(\log n)$ to $O(\log\log n)$. The average search time, however, could at worst double. Most of the results presented in this chapter or the previous one can be improved by a constant factor by increasing the number of hashing choices per key. For example, Theorems 5.1 and 5.2 can be easily generalized for open addressing hashing schemes that use $d \ge 2$ linear probe sequences. Similarly, all the two-way linear probing algorithms we proposed here can be generalized to d-way linear probing schemes. The maximum unsuccessful search time will, then, be at most $d\,C \log_d \log n + O(d)$, where C is a constant depending on α. This means that the best worst-case performance is when $d = 3$ where the minimum of $d/\log d$ is attained. The average search time, on the other hand, could triple.

The performance of these algorithms can be further improved by using Vöcking's scheme LeftMC(n, m, d) with $d \geq 2$ hashing choices. The maximum unsuccessful search time, in this case, is at most $C \log\log n / \log \phi_d + O(d)$, for some constant C depending on α. This is minimized when $d = o(\log\log n)$, but we know that it cannot get better than $C \log_2 \log n + O(d)$, because $\lim_{d\to\infty} \phi_d = 2$.

Off-Line Open Addressing

One can also improve the performance by considering off-line two-way open addressing schemes. We suggest here an off-line scheme for inserting keys into cells by open addressing where each key has two initial hashing cells chosen independently and uniformly at random, with replacement. We assume that all the initial hashing cells available for the keys are known in advance, i.e., before the algorithm starts. The algorithm is useful, therefore, for static open addressing hashing. We recall first some of the results about off-line two-way chaining obtained in Chapter 3, in particular, the bounds on the k-orientability threshold c_k, for $k \geq 2$. We have recorded that c_k/k converges exponentially to 1; indeed, for k large enough, we have

$$1 - 2^k \exp\left(-k + 1 + e^{-k/4}\right) < c_k/k < 1 - \exp\left(-2k\left(1 - e^{-2k}\right)\right) .$$

We also bounded c_k for small k, for example, we know that $c_2 \geq 1.67545943...$, $c_3 \geq 2.61845509...$, and $c_4 \geq 3.65354252....$ See Tables 3.3 and 3.5 for more of these bounds. Furthermore, algorithm Orient finds a k-orientation in $O(n^2)$ worst-case time, which can be improved to $O(n^{3/2})$ worst-case running time if we use Hopcroft and Karp's algorithm [89] for computing maximum matching in bipartite graphs. A $2k$-orientation can be found in linear time by algorithm AAR-Heuristic designed by Aichholzer et al. [3]. In the context of two-way chaining where each key chooses two chains independently and uniformly at random, with replacement, this means that if $\mu < c_k\nu$, where $\mu, \nu \in \mathbb{N}$, then one can hash off-line μ keys into a table with

ν chains by using one of these algorithms such that the longest chain has at most k elements, w.h.p.

Back to two-way open addressing hashing. Suppose that we would like to insert $m \in \mathbb{N}$ keys off-line into a hash table with $n \in \mathbb{N}$ cells where each key has two cells chosen independently and uniformly at random, with replacement. Each cell in the hash table can harbor at most one key. Moreover, suppose that the load factor $\alpha = m/n \in (0, c_k/k)$, for some fixed constant integer $k \geq 2$. Then one can assign the keys such that the maximum search time is at most $2k$, w.h.p. This can be done as follows. Divide the hash able into blocks of size k, assuming without loss of generality, that $\nu := n/k$ is an integer. So each key will have two blocks, and since the cells are chosen independently and uniformly at random, then so are the blocks. Since $m < c_k n/k = c_k \nu$, we can apply Hopcroft and Karp's algorithm to assign the keys to the ν blocks such that no block receives more than k keys, w.h.p. Within each block we are free to insert the keys in any fashion we like. Of course, there is a small probability that we may have more than k keys in a certain block but when this occurs, we try to insert these keys into the neighbor blocks.

Nonetheless, it is safe to say that w.h.p., the algorithm succeeds to place the keys such that the maximum load among all blocks is at most k. In other words, w.h.p., every key is inserted into a cell that belongs to one of its chosen blocks. Thus, the maximum search time is at most $2k$ as we have to search two blocks for each key. That is, if $k = 2$, for example, and $\alpha \in (0, 0.83772971...)$, then the maximum search time is at most 4, w.h.p. The only drawback is that the amortized insertion time is $O(\sqrt{n})$. This can be avoided, if we widen the blocks to be of size $2k$, and use the linear time algorithm AAR-Heuristic to find a $2k$-assignment, assuming of course that $\alpha \in (0, c_k/(2k))$. With this modification, the amortized insertion time is constant. The trade-off, however, is that α has to be small enough, and the maximum search time increases to $4k$. That is, if $k = 2$, and $\alpha \in (0,\ 0.41886485...)$, then the maximum search time is at most 8, w.h.p. Nevertheless, we know that c_k/k approaches one as k increases, and hence the performance of this scheme is still plausible. Thus, we have the following theorem.

Theorem 6.4. *Let $k \geq 2$ be a constant integer, and $\alpha \in (0, c_k/(2k))$. Suppose that we have $m = \alpha n$ keys to be inserted into a hash table of size n by open addressing, where each key has two initial cells chosen independently and uniformly at random, with replacement. Suppose also that all the initial hashing cells are known in advance. Then there is a linear time algorithm for inserting the keys such that maximum search time is $4k$, w.h.p.*

6.5 Simulation Results

We simulated the following linear probing algorithms we discussed in this chapter and Chapter 5 with the FCFS replacement strategy: the two-way linear probing algorithms ShortSeq, SmallCluster, WalkFirst, and DecideFirst, the locally linear algorithm LocallyLinear, and ClassicLinear (the classical linear probing algorithm which uses only one linear probe sequence). For each $n \in \{2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}\}$, and constant $\alpha \in \{0.4, 0.9\}$, we ran 100 simulations of each algorithm where we inserted $\lfloor \alpha n \rfloor$ keys into a hash table with n cells. We computed the average maximum cluster size, i.e.,

$$\frac{1}{100} \sum_{i=1}^{100} \text{size of the largest cluster in the } i\text{-th trial}\,,$$

and the average cluster size averaged over the 100 trials, that is,

$$\frac{1}{100} \sum_{i=1}^{100} \frac{m}{\text{number of clusters in the } i\text{-th trial}}\,.$$

The results are recorded in Tables 6.1 and 6.2 where the best worst-case performances are drawn in boldface, and the average cluster sizes are in italic.

Table 6.1 contains the simulation results of algorithms ClassicLinear, ShortSeq, and SmallCluster. It is evident that the average and the worst-case performances of SmallCluster and ShortSeq are better than ClassicLinear. Algorithm SmallCluster seems to have the best worst-case performance among the three algorithms. This is not a total surprise to us, because the algorithm considers more information (relative to the other two) before it makes its decision of where to insert the keys. It is also clear that there is a nonlinear increase, as a function of n, in the difference between the performances of these algorithms. This may suggest that the size of the largest cluster produced by algorithms ShortSeq and SmallCluster is roughly of the order of $\log n$.

n	α	ClassicLinear		ShortSeq		SmallCluster	
2^8	0.4	*2.02*	8.32	*1.76*	6.05	*1.76*	**5.90**
	0.9	*15.10*	87.63	*12.27*	50.19	*12.26*	**43.84**
2^{12}	0.4	*2.03*	14.95	*1.75*	9.48	*1.75*	**9.05**
	0.9	*15.17*	337.22	*12.35*	106.24	*12.34*	**78.75**
2^{16}	0.4	*2.02*	22.54	*1.75*	12.76	*1.75*	**12.08**
	0.9	*15.16*	678.12	*12.36*	155.26	*12.36*	**107.18**
2^{20}	0.4	*2.02*	29.92	*1.75*	16.05	*1.75*	**15.22**
	0.9	*15.17*	1091.03	*12.35*	203.16	*12.35*	**136.19**
2^{24}	0.4	*2.02*	38.00	*1.75*	19.54	*1.75*	**18.09**
	0.9	*15.17*	1514.80	*12.35*	253.93	*12.35*	**164.06**

Table 6.1: The average maximum cluster size and the average cluster size (in italic) over 100 simulations of the algorithms. The best performances are drawn in boldface.

The simulation data of algorithms LocallyLinear, WalkFirst, and DecideFirst are presented in Table 6.2. These algorithm are simulated with blocks of size $\lfloor (1-\alpha)^{-1}(\log_2 \log n + c) \rfloor$ for constant $c \in \{-1, 0, 1\}$. The purpose of this is to show that, practically, the additive and the multiplicative constants appearing in the definitions of the block sizes stated in Theorems 6.1, 6.2 and 6.3 can be chosen to be small. The hash table is partitioned into equal-sized blocks, except possibly the last one. The results show that algorithms LocallyLinear and WalkFirst have their best performances when $c = 0$. This is not really apparent in the performance of algorithm DecideFirst, and we are not sure of the actual reason. The performance of WalkFirst appears to be very close to that of LocallyLinear. This supports the conjecture that Theorem 6.3 is, in fact, true for any constant load factor $\alpha \in (0, 1)$, and the maximum unsuccessful search time of WalkFirst is at most $4(1-\alpha)^{-1} \log_2 \log n + O(1)$, w.h.p. The worst-case performance of algorithm DecideFirst seems to be close to the other ones when α is small; but it almost doubles when α is large. This may suggest that the multiplicative constant in the maximum unsuccessful search time established in Theorem 6.2 could be improved.

Comparing the simulation data from both tables, one can see that WalkFirst and LocallyLinear are superior to the others in worst-case performance. Surprisingly, the worst-case performances of algorithms SmallCluster and DecideFirst are very close, although, it appears that the difference becomes larger, as n increases.

		LocallyLinear		WalkFirst			DecideFirst		
n	α	$c=0$	$c=1$	$c=-1$	$c=0$	$c=1$	$c=-1$	$c=0$	$c=1$
2^8	0.4	*1.57* **4.34**	*1.62* 4.43	*1.61* 5.89	*1.65* **4.70**	*1.69* 4.86	*1.60* 5.21	*1.63* **4.81**	*1.67* 5.02
	0.9	*12.18* **33.35**	*12.78* 40.18	*11.55* 35.90	*12.54* **34.40**	*13.14* 41.15	*12.72* **43.06**	*13.48* 47.76	*14.07* 63.53
2^{12}	0.4	*1.62* **6.06**	*1.66* 6.26	*1.62* 7.10	*1.68* **6.32**	*1.71* 7.74	*1.61* 7.32	*1.68* **6.82**	*1.71* 6.97
	0.9	*12.42* **48.76**	*13.10* 60.68	*12.05* 53.29	*12.78* **51.80**	*13.19* 64.07	*13.01* **75.20**	*13.45* 94.98	*13.67* 126.82
2^{16}	0.4	*1.62* **7.14**	*1.68* 8.02	*1.65* 8.21	*1.68* **7.31**	*1.73* 8.30	*1.64* 9.05	*1.68* 8.92	*1.73* **8.78**
	0.9	*12.66* **59.61**	*13.13* 73.43	*12.38* 66.20	*12.98* **62.24**	*13.38* 76.94	*13.18* **105.61**	*13.53* 125.40	*13.79* 156.84
2^{20}	0.4	*1.65* **8.25**	*1.68* 8.95	*1.65* 9.85	*1.71* **8.50**	*1.73* 9.17	*1.64* 11.10	*1.71* 10.76	*1.73* **10.38**
	0.9	*12.83* **67.23**	*13.26* 83.02	*12.58* 76.82	*13.11* **69.45**	*13.45* 85.37	*13.30* **133.37**	*13.62* 145.30	*13.83* 157.07
2^{24}	0.4	*1.65* **9.05**	*1.71* 10.08	*1.68* 9.94	*1.71* **9.08**	*1.75* 10.34	*1.68* 12.22	*1.71* **12.05**	*1.75* 12.20
	0.9	*12.98* **74.02**	*13.35* 90.31	*12.77* 85.16	*13.23* **75.59**	*13.54* 92.20	*13.41* **159.95**	*13.69* 177.52	*13.89* 190.92

Table 6.2: The average maximum cluster size and the average cluster size (in italic) over 100 simulations of the algorithms with blocks of size $\lfloor (1-\alpha)^{-1}(\log_2 \log n + c) \rfloor$. The best performances are drawn in boldface.

Conclusion and Future Work

The worst-case performance of two-way hashing schemes with chaining and open addressing is studied and analyzed under different assumptions. The study, in short, demonstrates that two-way hashing is certainly worth considering in implementing dictionaries for the amazing worst-case performance it guarantees. One, on the other hand, should also consider the trade-offs that we have seen in this thesis, between space consumption (of chaining and open addressing), the expected and worst-case times of any operation, the number of hash functions, the randomness assumptions, and the simplicity of the algorithms. In the end, it remains in the hands of the practitioners who will judge which hashing scheme is best adapted to their application. These schemes, as well as other ones suggested by many researchers we mentioned in the thesis, should be put into hard applications.

On-line Two-way Chaining: The waiting time and the witness tree methods are used to reprove that worst-case search time of on-line uniform two-way chaining algorithm is $\log_2 \log n \pm O(1)$, asymptotically almost surely, for a hash table of constant load factor. The bounds are, then, extended to the fixed density model where the two independent hash functions behave according to fixed densities defined on the unit interval. The lower bound is shown to be true for any arbitrary fixed densities, while the matching upper only holds for bounded ones. Bounds for other cases such as the heavily- and lightly-loaded cases, or the dynamic case are also given. There are many directions that one can follow to extend this research. Most interestingly, are the following questions.

1. Is it possible to prove the lower bound uniformly over all densities for every $n \in \mathbb{N}$, while the two hash functions are still independent?

2. Can the upper bound for unbounded densities be improved?

3. In the classical uniform hashing with chaining, the chain length has the binomial distribution $\mathrm{Bin}(m, 1/n)$, and when the load factor $\alpha = m/n$ is constant, its limit distribution is $\mathrm{Poisson}(\alpha)$. What is the distribution of the chain length of on-line uniform two-way chaining, and does it have a limit distribution?

4. Theorem 0.2 states that on-line uniform two-way chaining is optimal among all on-line algorithms that use two independent truly uniform hash functions and do not reallocate the keys. Is it possible to improve the performance of on-line two-way chaining by using efficient adaptive reallocation techniques?

Off-line Two-way Chaining: A relationship between the off-line version of uniform two-way chaining and the k-orientability of random graphs is established. The k-orientability threshold c_k is tightly estimated and proved to be asymptotic to k. Algorithms for finding k-orientations are presented. This area of research can be broadened as follows.

1. Is there an elegant short proof for bounding c_k from below?

2. The best known lower bound on c_2 is 1.67545943.... Is it tight?

3. Is there a linear time algorithm for finding a k-orientation for $k \geq 2$?

4. Can one put the threshold bounds into an effective use to design efficient realistic hashing schemes with reasonable worst-case insertion time and plausible worst-case search time?

5. Is it possible to generalize the k-orientability study for nonuniform random graphs where the vertices are chosen according to different probabilities? This is related to off-line nonuniform two-way chaining.

Two-way Linear Probing: The concept of two-way linear probing is suggested as a translation of two-way chaining paradigm to open addressing hashing. An $\Omega(\log \log n)$ universal lower bound is proved on the worst-case performance of any two-way linear probing algorithm. Unfortunately, two simple two-way linear probing algorithms are proved to yield unsatisfactory performances. Alternatively, two other efficient algorithms that have matching upper bounds on their worst-case performances are proposed. Simulation outputs that confirm the theoretical results are also provided. Some of the possible future works in this area can be summarized as follows.

1. The worst-case performance of algorithm WalkFirst is proved only for load factor $\alpha < 1/2$, although, the simulation results suggest that it may be true for any $\alpha \in (0, 1)$. Can the proof of Theorem 6.3 be extended for $\alpha \in (0, 1)$?

2. Is it possible to improve the multiplicative constant factors appearing in the upper bounds on the performances of algorithms WalkFirst and DecideFirst?

3. How does (increasing or decreasing) the block size affect the performances of the algorithms designed in Chapter 6?

4. Is there a two-way linear probing algorithm whose worst-case search time is at most $c \log \log n + \xi(\alpha)$, for some function ξ, and constant c that is not an increasing function of α?

5. What is the upper bound on the worst-case performance of any algorithm that satisfies the conditions of Theorem 5.3? The same question can be applied to Theorem 5.4.

6. Is there an algorithm that satisfies the conditions of Theorem 5.3, and has an $O(\sqrt{\log n / \log \log n})$ upper bound on its worst-case performance?

> *And do not say of anything: I am doing that tomorrow. Unless Allah pleases; and remember your Lord when you forget, and say: maybe my Lord will guide me to a nearer course of truth than this.*
>
> THE NOBLE QUR'AN, (18: 23–24)

Appendix: Finishing the Proof of Theorem 3.4

Recall that $b = 0.772907804...$, and

$$\beta_3 = 2.61845509\ldots, \quad \beta_4 = 3.65354252\ldots, \quad \beta_5 = 4.71959504\ldots.$$

We need to prove that $t(k, \beta_k, p)$ is an increasing function in p on $[e^{-3}, b]$, and $h(k, \beta_k, p)$ is a decreasing function in p on $[b, \beta_k/k]$. Moreover, we shall prove that the following two conditions are satisfied:

$$\max_{e^{-3} \le p \le b/2} g(k, \beta, p) < 2^{-b/2}, \quad \max_{b/2 \le p \le b} g(k, \beta, p) < 2^{-b}, \tag{6.3}$$

where

$$\begin{aligned}
t(k, \beta, p) &= \left(\frac{2\beta(1-p^2)}{2\beta - (b+p)k}\right)^{\beta - (b+p)k/2} \left(\frac{2\beta p^2}{(b+p)k}\right)^{(b+p)k/2}, \\
g(k, \beta, p) &= f(k, \beta, p) \exp\left(\frac{-kp(b - \log b - 1)}{k+1}\right), \\
f(k, \alpha, p) &= \left(\frac{\alpha(1-p^2)}{\alpha - kp}\right)^{\alpha - kp} \left(\frac{\alpha p}{k}\right)^{kp},
\end{aligned}$$

and

$$h(k, \alpha, p) = \begin{cases} 1, & \text{if } p = 0; \\ p^{-p}(1-p)^{p-1} f(k, \alpha, p), & \text{for } p \in (0, \alpha/k); \\ (\alpha/k)^{2\alpha}, & \text{if } p = \alpha/k. \end{cases}$$

We first prove the monotonicity of the function t.

Lemma 6.3. *For all $k = 3, 4, 5$, the function $t(k, \beta_k, p)$ is a strictly increasing function in p on $[e^{-3}, b]$.*

Proof. Notice that for $p \in [e^{-3}, b]$,

$$\begin{aligned}\log t(k, \beta_k, p) \;&=\; \left(\beta_k - \frac{(b+p)k}{2}\right)\left(\log(1-p^2) - \log\left(1 - \frac{(b+p)k}{2\beta_k}\right)\right) \\ &\quad + \frac{(b+p)k}{2}\left(2\log p - \log(b+p) + \log\frac{2\beta_k}{k}\right).\end{aligned}$$

We shall prove that the derivative of $\log t(k, \beta_k, p)$ is positive, for all $p \in [e^{-3}, b]$, which will imply that t is an increasing function. So, consider the following:

$$\begin{aligned}\frac{d}{dp}(\log t) \;&=\; -\frac{k}{2}\log(1-p^2) - \frac{(2\beta_k - bk)p - kp^2}{1-p^2} + \frac{k}{2}\log\left(1 - \frac{(b+p)k}{2\beta_k}\right) \\ &\quad + k\log p - \frac{k}{2}\log(b+p) + \frac{k}{2}\log\frac{2\beta_k}{k} + \frac{bk}{p} + k \\ &> \; \frac{k}{2}\log\left(1 - \frac{(b+p)k}{2\beta_k}\right) - \frac{(2\beta_k - bk)p - kp^2}{1-b^2} + k\log p + \frac{bk}{p} \\ &\quad + \frac{k}{2}\log\frac{\beta_k}{bk} + k \\ &\stackrel{\text{def}}{=}\; \varphi_k(p),\end{aligned}$$

because $p < b < \beta_k/k$. Hence,

$$\begin{aligned}\varphi_k'(p) \;&=\; -\frac{k^2}{4\beta_k - 2(b+p)k} - \frac{2\beta_k - bk - 2kp}{1-b^2} + \frac{k}{p} - \frac{bk}{p^2} \\ &<\; -\frac{k^2}{4\beta_k - 2(b+p)k} - \frac{2\beta_k - bk - 2kp}{1-b^2} \\ &<\; \frac{-k^2(1+b^2) - 8\beta_k(\beta_k - bk) + 6kp(2\beta_k - bk) - 4k^2p^2}{(4\beta_k - 2(b+p)k)(1-b^2)} \\ &\stackrel{\text{def}}{=}\; \frac{\xi_k(p)}{(4\beta_k - 2(b+p)k)(1-b^2)},\end{aligned}$$

which is negative if and only if the numerator $\xi_k(p)$ is. Now if $k = 4$ or 5, the numerator attains its maximum on $[0, b]$ at b, because

$$\xi_k'(p) = 6k(2\beta_k - bk) - 8k^2p = 0 \iff p = \frac{3}{4k}(2\beta_k - bk) > 0.79 > b,$$

and $\xi_k(b) = \xi_k(0) + 2bk(6\beta_k - 5bk) > \xi_k(0)$. However, $\xi_4(b) = -2.0188808...$, and $\xi_5(b) = -2.6967152....$ On the other hand, if $k = 3$, the numerator $\xi_3(p)$ has a maxima at $\hat{p} = (2\beta_3 - 3b)/4 = 0.729546692...$; and $\xi_3(0) < \xi_3(b) < \xi_3(\hat{p}) = -1.4945682....$ Consequently, $\varphi_k'(p) < 0$, for all $k = 3, 4, 5$, and thence, $\varphi_k(p)$ is a decreasing function on $[e^{-3}, b]$. Thus, $\varphi_k(p) > \varphi_k(b) > 0$, for all $k = 3, 4, 5$, because $\varphi_3(b) = 1.00758127...$, $\varphi_4(b) = 1.40196584...$, and $\varphi_5(b) = 1.65799854....$ This completes the proof. □

Next we prove the monotonicity of the function h.

Lemma 6.4. *For all $k = 3, 4, 5$, the function $h(k, \beta_k, p)$ is a strictly decreasing function in p on $[b, \beta_k/k]$.*

Proof. For $p \in [b, \beta_k/k]$, let

$$\begin{aligned} \varphi_k(p) &:= \log h(k, \beta_k, p) \\ &= (\beta_k - kp)\log(1-p^2) - (\beta_k - kp)\log(\beta_k - pk) + (k-1)p\log p \\ &\quad -(1-p)\log(1-p) - kp\log k + \beta_k \log \beta_k\,. \end{aligned}$$

It suffices to prove that $\varphi_k'(p) < 0$. Notice that

$$\begin{aligned} \varphi_k'(p) &= -k\log(1-p^2) - \frac{2p(\beta_k - kp)}{1-p^2} + k\log(\beta_k - kp) + (k-1)\log p \\ &\quad + \log(1-p) + 2k - k\log k\,. \end{aligned}$$

We shall prove that $\varphi_k'(p)$ is a decreasing function on $[b, \beta_k/k]$, for all $k = 3, 4, 5$, so that $\varphi_k'(p) < \varphi_k'(b) < 0$, because $\varphi_3'(b) = -1.32950101...$, $\varphi_4'(b) = -0.62432743...$, and $\varphi_5'(b) = -0.07696858....$ For that we show $\varphi_k''(p) < 0$, as follows:

$$\begin{aligned} \varphi_k''(p) &= \frac{2kp}{1-p^2} - \frac{2\beta_k - 4kp + 2\beta_k p^2}{(1-p^2)^2} - \frac{k^2}{\beta_k - pk} + \frac{k-1}{p} - \frac{1}{1-p} \\ &= \frac{\xi_k(p)}{p(\beta_k - pk)(1-p^2)^2}\,, \end{aligned}$$

where

$$\begin{aligned}\xi_k(p) &= \beta_k k - \beta_k - (2k^2 - k + 2\beta_k^2 + \beta_k)p + (6k\beta_k + k + \beta_k)p^2 \\ &\quad -(2k^2 + k + 2\beta_k^2 - \beta_k)p^3 + k(\beta_k - 1)p^4 .\end{aligned}$$

So we need to prove that $\xi_k(p) < 0$, on $[b, \beta_k/k]$. Consider the following:

$$\begin{aligned}\xi_k'(p) &= -2k^2 + k - 2\beta_k^2 - \beta_k + 2(6k\beta_k + k + \beta_k)p - 3(2k^2 + k + 2\beta_k^2 - \beta_k)p^2 \\ &\quad +4k(\beta_k - 1)p^3 ; \\ \xi_k''(p) &= 2(6k\beta_k + k + \beta_k) - 6(2k^2 + k + 2\beta_k^2 - \beta_k)p + 12k(\beta_k - 1)p^2 ,\end{aligned}$$

and hence,

$$\begin{aligned}\xi_k'''(p) &= -6(2k^2 + k + 2\beta_k^2 - \beta_k) + 24k(\beta_k - 1)p \\ &< -12k^2 - 6k - 12\beta_k^2 + 6\beta_k + 24\beta_k^2 - 24\beta_k \\ &= -12(k^2 - \beta_k^2) - 6k - 18\beta_k < 0 ,\end{aligned}$$

for all $k = 3, 4, 5$. This means that $\xi_k'(p)$ is a concave function on $[b, \beta_k/k]$. Computing the values of ξ_k' at both of the endpoints of the interval $[b, \beta_k/k]$, for all $k = 3, 4, 5$, we get

$$\xi_3'(b) = 1.66122202... , \quad \xi_4'(b) = 2.8140136... , \quad \xi_5'(b) = 4.0235492... ,$$

$$\xi_3'(\beta_3/3) = 0.3170448... , \quad \xi_4'(\beta_4/4) = 0.3895055... , \quad \xi_5'(\beta_5/5) = 0.4653657... .$$

Since they are all positive, then $\xi_k'(p) > 0$ on $[b, \beta_k/k]$. Consequently, $\xi_k(p)$ is an increasing function on the interval $[b, \beta_k/k]$. Therefore, $\xi_k(p) \le \xi_k(\beta_k/k) < 0$, for all $k = 3, 4, 5$, because $\xi_3(\beta_3/3) = -0.44566294...$, $\xi_4(\beta_4/4) = -0.40138310...$, and $\xi_5(\beta_5/5) = -0.2804545....$ □

The next lemma proves that first condition in (6.3) is satisfied.

Lemma 6.5. *For $k = 3, 4, 5$, the following condition is satisfied:*

$$\max_{e^{-3} \le p \le b/2} g(k, \beta_k, p) < 2^{-b/2}\,.$$

Proof. By inequalities (3.4) and (3.5), we have

$$\begin{aligned}
\log g(k, \beta_k, p) &= (\beta_k - kp)\log(1-p^2) - (\beta_k - kp)\log(1 - kp/\beta_k) \\
&\quad + kp \log \frac{\beta_k p}{k} - \frac{kp\,(b - \log b - 1)}{k+1} \\
&< kp - \beta_k p^2 + kp^3 + kp \log \frac{\beta_k p}{k} - \frac{kp\,(b - \log b - 1)}{k+1} \overset{\text{def}}{=} \varphi_k(p)\,.
\end{aligned}$$

However,

$$\begin{aligned}
\varphi_k'(p) &= 2k - 2\beta_k p + 3kp^2 + k \log \frac{\beta_k p}{k} - \frac{k(b - \log b - 1)}{k+1}\,; \\
\varphi_k''(p) &= -2\beta_k + 6kp + k/p\,.
\end{aligned}$$

One can see that $\varphi_k''(p)$ attains its minimum value on $[e^{-3}, b/2]$ at the point $p = \beta_k/(6k)$, that is, $\varphi_k''(p) \ge -\beta_k + 6k^2/\beta_k > -k + 6k > 0$, for all $k = 3, 4, 5$. This means that $\varphi_k(p)$ is a convex function on $[e^{-3}, b/2]$. Evaluating $\varphi_k(p)$ at the endpoints $p = e^{-3}$ and $p = b/2$, for all $k = 3, 4, 5$, we get

$$\varphi_3(e^{-3}) = -0.32629896\ldots\,, \quad \varphi_4(e^{-3}) = -0.42611632\ldots\,, \quad \varphi_5(e^{-3}) = -0.52458518\ldots\,,$$

$$\varphi_3(b/2) = -0.32735014\ldots\,, \quad \varphi_4(b/2) = -0.38811609\ldots\,, \quad \varphi_5(b/2) = -0.44244344\ldots\,.$$

Thus, we conclude that $\varphi_k(p) < -(b/2)\log 2 = -0.267869432...$, for all $p \in [e^{-3}, b/2]$, and $k = 3, 4, 5$, which ends the proof. ∎

Before we verify the second condition in (6.3) we need to establish two lemmas. The following bounds can be proved easily by finding the maximum of the polynomials using the first derivative test.

Lemma 6.6. *The following bounds hold for all $p \in [0.38, 0.78]$:*

1. $1.674 - 10.66p + 22.093p^2 - 15p^3 < 0$.

2. $-9.62 + 47.84p - 77.571p^2 + 40.66p^3 < 0$.

3. $3 - 19.2p + 37.93p^2 - 24p^3 < 0$.

4. $-20.77 + 92.92p - 137.53p^2 + 67.2p^3 < 0$.

5. $4.84 - 30.4p + 58.0372p^2 - 35p^3 < 0$.

6. $-36.5119 + 153.784p - 215.4634p^2 + 100.4p^3 < 0$.

Lemma 6.7. *For $k = 3, 4, 5$, and $p \in [e^{-3}, \beta_k/k]$, define the function*

$$\varphi_k(p) := (\beta_k - kp)\log\frac{\beta_k(1-p^2)}{\beta_k - kp}.$$

Then $\varphi_k''(p) \leq -k-2$, for all $p \in [0.38, 0.78]$.

Proof. Notice that

$$\varphi_k''(p) = \frac{-k^2 - 2\beta_k^2 + 8\beta_k kp - (4k^2 + 2\beta_k^2)p^2 + k^2p^4}{(1-p^2)^2\,(\beta_k - kp)}.$$

Thus, $\varphi_k''(p) \leq -k-2$ if and only if

$$\begin{aligned}
0 &\geq -k^2 - 2\beta_k^2 + 8\beta_k kp - (4k^2 + 2\beta_k^2)p^2 + k^2p^4 + (k+2)(1-p^2)^2\,(\beta_k - kp) \\
&= -k^2 - 2\beta_k^2 + (k+2)\beta_k + k(8\beta_k - k - 2)p - 2(2k^2 + \beta_k^2 + (k+2)\beta_k)p^2 \\
&\quad +2k(k+2)p^3 + (k^2 + (k+2)\beta_k)p^4 - k(k+2)p^5 \\
&\stackrel{\text{def}}{=} \psi_k(p).
\end{aligned}$$

We prove that $\psi_k \leq 0$, for $k = 3, 4, 5$, by bounding ψ_k from above by a polynomial of degree 3 for which we can easily find its maximum values. For $k = 3$, we get

$$\begin{aligned}
\psi_3(p) &< -9.62 + 47.84p - 75.897p^2 + 30p^3 + 22.093p^4 - 15p^5 \\
&< -9.62 + 47.84p - 75.897p^2 + 30p^3 + 10.66p^3 - 1.674p^2 \\
&= -9.62 + 47.84p - 77.571p^2 + 40.66p^3 \;<\; 0,
\end{aligned}$$

where we have used bounds (1) and (2) of Lemma 6.6. Similarly, for $k = 4$, we get

$$\begin{aligned} \psi_4(p) &< -20.77 + 92.92p - 134.53p^2 + 48p^3 + 37.93p^4 - 24p^5 \\ &< -20.77 + 92.92p - 134.53p^2 + 48p^3 + 19.2p^3 - 3p^2 \\ &= -20.77 + 92.92p - 137.53p^2 + 67.2p^3 \ < \ 0 \,, \end{aligned}$$

where we have used the bounds (2) and (3) of Lemma 6.6. Finally, for $k = 5$, and using the last two bounds in Lemma 6.6, we obtain

$$\begin{aligned} \psi_5(p) &< -36.5119 + 153.784p - 210.6234p^2 + 70p^3 + 58.0372p^4 - 35p^5 \\ &< -36.5119 + 153.784p - 210.6234p^2 + 70p^3 + 30.4p^3 - 4.84p^2 \\ &< -36.5119 + 153.784p - 215.4634p^2 + 100.4p^3 \ < \ 0\,. \end{aligned}$$

□

Now we are ready to verify the second condition in (6.3).

Lemma 6.8. *For $k = 3, 4, 5$, the following condition is satisfied:*

$$\max_{b/2 \le p \le b} g(k, \beta_k, p) < 2^{-b}\,.$$

Proof. Recall that

$$\log g(k, \beta_k, p) = (\beta_k - kp) \log \frac{1 - p^2}{1 - kp/\beta_k} + kp \log \frac{\beta_k p}{k} - \frac{kp\,(b - \log b - 1)}{k + 1}\,.$$

For convenience, let

$$\varphi_k(p) := (\beta_k - kp) \log \frac{1 - p^2}{1 - kp/\beta_k}\,.$$

By Taylor's theorem, we know that for $p \in [b/2, b]$, there exists a point $x \in [b/2, b]$ such that

$$\begin{aligned} \varphi_3(p) &= \varphi_3(0.7) + \varphi_3'(0.7)(p - 0.7) + \varphi_3''(x)(p - 0.7)^2/2 \\ &< 0.491 - 1.26(p - 0.7) - 2.5(p - 0.7)^2 < 0.15 + 2.24p - 2.5p^2\,, \end{aligned}$$

where we have used Lemma 6.7. Similarly, we expand φ_4, and φ_5 about the point $p = 0.6$ to obtain the following estimations:

$$\begin{aligned}\varphi_4(p) &< 0.782 - 0.84(p-0.6) - 3(p-0.6)^2 < 0.21 + 2.76p - 3p^2\,,\\ \varphi_5(p) &< 0.97 - (p-0.6) - 3.5(p-0.6)^2 - 3.5(p-0.6)^2 < 0.31 + 3.2p - 3.5p^2\,.\end{aligned}$$

Also, by Taylor's theorem, there exists a point $y \in [b/2, b]$ such that

$$\begin{aligned}\log p &= \log(0.75) + \frac{1}{0.75}(p-0.75) - \frac{1}{2y^2}(p-0.75)^2\\ &< -0.287 + 1.34(p-0.75) - 0.83(p-0.75)^2\\ &= -1.758 + 2.585p - 0.83p^2\,.\end{aligned}$$

Therefore, putting these bounds together, we see that for $p \in [b/2, b]$,

$$\begin{aligned}\log g(3,\beta_3,p) &< 0.15 - 3.464p + 5.255p^2 - 2.49p^3 < -0.536\\ \log g(4,\beta_4,p) &< 0.21 - 4.658p + 7.34p^2 - 3.32p^3 < -0.536\\ \log g(5,\beta_5,p) &< 0.31 - 5.903p + 9.425p^2 - 4.15p^3 < -0.536\,.\end{aligned}$$

The last inequalities can be proved by finding the maximum value using the first derivative test. Notice that $-0.536 < -b\log 2 = -0.535738865...$, and this ends the proof. $\square$

List of Algorithms

Index of Notation

SETS AND NUMBERS

ASYMPTOTICS

PROBABILITY

HASHING

Graphs

Bibliography

[1] M. Adler, P. Berenbrink, and K. Schroeder, "Analyzing an infinite parallel job allocation process," in: *Proceedings of the European Symposium on Algorithms*, pp.417–428, 1998.

[2] M. Adler, S. Chakrabarti, M. D. Mitzenmacher, and L. Rasmussen, "Parallel randomized load balancing," in: *Proceedings of the 27th ACM Symposium on Theory of Computing (STOC)*, pp. 238–247, 1995.

[3] O. Aichholzer, F. Aurenhammer, and G. Rote, "Optimal graph orientation with storage applications," SFB-Report F003-51, SFB 'Optimierung und Kontrolle', TU-Graz, Austria, 1995.

[4] D. Aldous, "Hashing with linear probing, under non-uniform probabilities," *Probability in the Engineering and Informational Sciences*, vol. 2, pp. 1–14, 1988.

[5] N. Alon, L. Babai and A. Itai, "A fast and simple randomized parallel algorithm for the maximal independent set problem," *Journal of Algorithms*, vol. 7 (4), pp. 567–583, 1986.

[6] N. Alon, M. Dietzfelbinger, P. B. Miltersen, E. Petrank, and G. Tardos, "Linear hash functions," *Journal of the ACM*, vol. 46 (5), pp. 667–683, 1999. A

preliminary version appeared in: *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pp. 465–474, 1997.

[7] N. Alon and J. H. Spencer, *The Probabilistic Method*, 2nd ed., John Wiley, New York, 2000.

[8] A. Anagnostopoulos, I. Kontoyiannis, and E. Upfal, "Steady state analysis of balanced-allocation routing," submitted, 2002. A preliminary version available at http://www.dam.brown.edu/people/yiannis/PAPERS/balanced.ps.

[9] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman, "Sorting in linear time?," in: *Proceedings of the 27th ACM Symposium on Theory of Computing (STOC)*, pp. 427–436, 1995.

[10] A. Andersson, "Faster deterministic sorting and searching in linear space," in: *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 135–141, 1996.

[11] D. Angluin and L. G. Valiant, "Fast probabilistic algorithms for hamiltonian paths and matchings," *Journal of Computer and Systems Science*, vol. 18, pp. 155–193, 1979.

[12] Y. Azar, A. Z. Broder, and A. R. Karlin, "On-line load balancing," in: *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 218–225, 1992.

[13] Y. Azar, A. Z. Broder, A. R. Karlin and E. Upfal, "Balanced allocations," *SIAM Journal on Computing*, vol. 29 (1), pp. 180–200, 2000. A preliminary version of this paper appeared in *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC)*, pp. 593–602, 1994.

[14] Y. Azar, J. Noar, and R. Rom, "The competitiveness of on-line assignments," in: *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 203–210, 1992.

[15] G. de Balbine, *Computational Analysis of the Random Components Induced by Binary Equivalence Relations*, Ph.D. thesis, California Institute of Technology, 1969.

[16] G. Bennett, "Probability inequalities for the sum of independent random variables," *Journal of the American Statistical Association*, vol. 57, pp. 33–45, 1962.

[17] P. Berenbrink, A. Czumaj, T. Friedetzky, and N. D. Vvedenskaya, "Infinite parallel job allocations," in: *Proceedings of the 11th ACM Symposium on Parallel Algorithms and Architectures*, pp. 99–108, Bar Harbor, Maine, July 9–13, 2000.

[18] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, "Balanced allocations: the heavily loaded case," in: *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC)*, pp. 745–754, 2000.

[19] P. Berenbrink , F. Meyer auf der Heide, K. Schröder, "Allocating weighted jobs in parallel," *Theory of Computing Systems*, pp. 281–300, 1999.

[20] S. Boucheron, F. Gamboa, and C. Léonard, "Bins and balls: large deviations of the empirical occupancy process," Rapport de recherche du LRI No. 1255, Université Paris-Sud, 2000.

[21] B. Bollobás, A. Z. Broder, and I. Simon, "The cost distribution of clustering in random probing," *Journal of the ACM*, vol. 37 (2), pp. 224–237, 1990.

[22] R. P. Brent, "Reducing the retrieval time of scatter storage techniques," *Communications of the ACM*, vol. 16 (2), pp. 105–109, 1973.

[23] A. Z. Broder and A. Karlin, "Multilevel adaptive hashing," in: *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 43–53, 2000.

[24] A. Broder and M. D. Mitzenmacher, "Using multiple hash functions to improve IP lookups," in: *Proceedings of the IEEE INFOCOM 2001 Conference*, Anchorage, Alaska USA , April 2001. Full version available as Technical Report TR–03–00, Department of Computer Science, Harvard University, Cambridge, MA, 2000.

[25] J. Buhler, "Efficient large-scale sequence comparison by locality-sensitive hashing," *Bioinformatics*, vol. 17 (5), pp. 419–428, 2001.

[26] J. Byers, J. Considine, and M. D. Mitzenmacher, "Simple load balancing for distributed hash tables," in: *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, pp. 80–87, 2003.

[27] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Science*, vol. 18, pp. 143–154, 1979.

[28] P. Celis, *Robin Hood Hashing*, Ph.D. thesis, Computer Science Department, University of Waterloo, 1986. Available also as Technical Report CS-86-14.

[29] P. Celis, P. Larson, and J. I. Munro, "Robin Hood hashing (preliminary report)," in: *Proceedings of the 26th Symposium on Foundations of Computer Science (FOCS)*, pp. 281–288, 1985.

[30] P. Chassaing and G. Louchard, "Phase transition for parking blocks, Brownian excursion and coalescence," *Random Structures Algorithms*, vol. 21 (1), pp. 76–119, 2002.

[31] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sums of observations," *Annals of Mathematical Statistics*, vol. 23, pp. 493–507, 1952.

[32] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky, "The bit extraction problem or t-resilient functions (preliminary version)," in: *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 396–407, 1985.

[33] R. Cole, A. Frieze, B. M. Maggs, M. D. Mitzenmacher, A. W. Richa, R. K. Sitaraman, and E. Upfal, "On balls and bins with deletions," in: *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, LNCS 1518, Springer-Verlag, pp. 145–158, 1998.

[34] R. Cole, B. M. Maggs, F. Meyer auf der Heide, M. D. Mitzenmacher, A. W. Richa, K. Schroeder, R. K. Sitaraman, and B. Voecking, "Randomized protocols for low-congestion circuit routing in multistage interconnection networks," in: *Proceedings of the 29th ACM Symposium on the Theory of Computing*, pp. 378–388, 1998.

[35] T. H. Cormen, C. L. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[36] Z. J. Czech, G. Havas, and B. S. Majewski, "Perfect hashing," *Theoretical Computer Science*, vol. 182 (1-2), pp. 1–143, 1997.

[37] A. Czumaj, F. Meyer auf der Heide, and V. Stemann, "Contention resolution in hashing based shared memory simulations," *SIAM Journal on Computing*, vol. 29, No. 5, pp. 1703–1739, 2000.

[38] A. Czumaj and V. Stemann, "Randomized allocation processes," in: *Proceedings of the 38th Symposium on Foundations of Computer Science (FOCS)*, pp. 194–203, 1997.

[39] A. Czumaj and V. Stemann, "Randomized allocation processes," *Random Structures and Algorithms*, vol. 18, Issue 4, pp. 297–331, June 2001.

[40] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions," abstract appeared in: *DIMACS Workshop on Streaming Data Analysis*, 2003.

[41] L. Devroye, "The expected length of the longest probe sequence for bucket searching when the distribution is not uniform," *Journal of Algorithms*, vol. 6, pp. 1–9, 1985.

[42] L. Devroye, *Lecture Notes on Bucket Algorithms*, Birkhäuser Verlag, Boston, 1986.

[43] L. Devroye and L. Györfi, *Nonparametric Density Estimation: The L1 View*, John Wiley, New York, 1985.

[44] L. Devroye and P. Morin, "Cuckoo hashing: further analysis," *Information Processing Letters*, vol. 86, pp. 215–219, 2004.

[45] L. Devroye, P. Morin, and A. Viola, "On worst-case Robin Hood hashing," to appear in: *SIAM Journal on Computing*, 2003.

[46] R. Diestel, *Graph Theory*, Springer-Verlag, Graduate Texts in Mathematics, vol. 173, New York, 1997.

[47] M. Dietzfelbinger, "Universal hashing and k-wise independent random variables via integer arithmetic without primes," in: *Proceedings of the 13th Symposium*

on Theoretical Aspects of Computer Science, LNCS 1046, Springer-Verlag, pp. 569–580, 1996.

[48] M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger, "Polynomial hash functions are reliable (extended abstract)," in: *Proceedings of the 19th International Colloquium on Automata Languages and Programming*, LNCS 623, Springer-Verlag, pp. 235–246, 1992.

[49] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen, "A reliable randomized algorithm for the closest-pair problem," *Journal of Algorithms*, vol. 25, pp. 19–51, 1997.

[50] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. Tarjan, "Dynamic perfect hashing: upper and lower bounds," *SIAM Journal on Computing*, vol. 23 (4), pp. 738–761, 1994. A preliminary version appeared in: *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 524–531, 1988.

[51] M. Dietzfelbinger and F. 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*, LNCS 443, Springer-Verlag, pp. 6–19, 1990.

[52] M. Dietzfelbinger and F. Meyer auf der Heide, "High performance universal hashing, with applications to shared memory simulations," in: *Data Structures and Efficient Algorithms*, LNCS 594, Springer-Verlag, pp. 250–269, 1992.

[53] M. Dietzfelbinger and P. Wolfel, "Almost random graphs with simple hash functions," in: *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pp. 629–638, 2003.

[54] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas, "Neighborhood preserving hashing and approximate queries," in: *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 251–259, 1994.

[55] E. Drinea, A. Frieze, and M. D. Mitzenmacher, "Balls and nins models with feedback," in: *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 308–315, 2002.

[56] D. Dubhashi, V. Priebe, and D. Ranjan, "Negative dependence through the FKG inequality," BRICS Report Series, RS–96–27, 1996.

[57] D. Dubhashi, and D. Ranjan, "Balls and bins: a study in negative dependence," *Random Structures and Algorithms*, vol. 13 (2), pp. 99–124, 1998.

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

[59] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Transactions on Software Engineering*, vol. 12, pp. 662–675, 1986.

[60] C. Engelmann and J. Keller, "Simulation-based comparison of hash functions for emulated shared memory," in: *Proceedings of the 5th International Conference on Parallel Architectures and Languages Europe*, LNCS 694, Springer-Verlag, pp. 1–11, 1993.

[61] P. Erdös and A. Rényi, "On the evolution of random graphs," *Publ. Math. Ins. Hunger. Acad. Sci.*, vol. 5, PP. 17–61, 1960.

[62] A. P. Ershov, "On programming of arithmetic statements," (in Russian), *Doklady Akademii Nauk SSSR*, vol. 118 (3), pp.427–430, 1958.

[63] J. D. Esary, F. Proschan, and D. W. Walkup, "Association of random variables, with applications," *Annals of Mathematical Statistics*, vol. 38, pp. 1466–1474, 1967.

[64] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problem," *SIAM Journal on Computing*, vol. 5 (4), pp. 691–703, 1976.

[65] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong, "Extendible hashing–a fast access method for dynamic files," *ACM Transactions on Database Systems*, vol. 4 (3), pp. 315–344, 1979.

[66] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1, John Wiley, New York, 1968.

[67] P. Flajolet, P. V. Poblete, and A. Viola, "On the analysis of linear probing hashing," *Algorithmica*, vol. 22, pp. 490–515, 1998.

[68] C. M. Fortuin, P. N. Kasteleyn, and J. Ginibre, "Correlation inequalities on some partially ordered sets." *Communications in Mathematical Physics*, vol. 22, pp. 89–103, 1971.

[69] D. Fotakis, R. Pagh, P. Sanders, and P. G. Spirakis, "Space efficient hash tables with worst case constant access time," in: *Proceedings of the 20th Symposium on Theoretical Aspects of Computer Science*, LNCS 2607, Springer-Verlag, pp. 271–282, 2003.

[70] N. Fountoulakis, *Thresholds and the Structure of Sparse Random Graphs*, Ph.D. thesis, University of Oxford, 2003.

[71] A. Frank, "On the orientation of graphs," *Journal of Combinatorial Theory*, Series B, vol. 28 (3), pp. 251–261, 1980.

[72] A. Frank, "Orientations of graphs and submodular flows," *Congressus Numerantium*, (A. J. W. Hilton, ed.), vol. 113, pp. 111–142, 1996.

[73] A. Frank and A. Gyárfás, "How to orient the edges of a graph," in: *Combinatorics*, Colloquia Mathematica Societatis János Bolyai, vol. 18, pp. 353–364, 1976.

[74] M. Fredman, J. Komlós, E. Szemerédi, "Storing a sparse table with $O(1)$ worst case access time," *Journal of the ACM*, vol. 31, pp. 538–544, 1984.

[75] D. Gardy , "Occupancy urn models in analysis of algorithms," *Journal of Statistical Planning and Inference*, vol. 101 (1–2), pp. 95–105, 2002.

[76] A. K. Garg and C. C. Gotlieb, "Order-preserving key transformations," *ACM Transactions on Database Systems*, vol. 11 (2), pp. 213–234, 1986.

[77] A. Gionis, P. Indyk, and R. Motwani, "Similarity searching in high dimensions via hashing," in: *Proceedings of the International Conference on Very Large Data Bases*, pp. 518–529, 1999.

[78] G. H. Gonnet, "Open addressing hashing with unequal-probability keys," *Journal of Computer and System Sciences*, vol. 20, pp. 354–367, 1980.

[79] G. H. Gonnet, "Expected length of the longest probe sequence in hash code searching," *Journal of the ACM*, vol. 28, pp. 289–304, 1981.

[80] G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures*, Addison-Wesley, Workingham, 1991.

[81] G. H. Gonnet and J. I. Munro, ''Efficient ordering of hash tables," *SIAM Journal on Computing*, vol. 8 (3), pp. 463–478, 1979.

[82] G. H. Gonnet, L. D. Rogers, J. A. George, "An algorithmic and complexity analysis of interpolation search," *Acta Informatica*, vol. 39, pp. 39–52, 1980.

[83] G. R. Grimmett and D. R. Stirzaker, *Probability and Random Processes*, Oxford University Press, Oxford, 2001.

[84] L. J. Guibas, "The analysis of hashing techniques that exhibit k-ary clustering," *Journal of the ACM*, vol. 25 (4), pp. 544–555, 1978.

[85] M. De Guzman, *Real Variable Methods in Fourier Analysis*, North-Holland, Amsterdam, 1981.

[86] T. Hagerup, P. B. Miltersen, and R. Pagh, "Deterministic dictionaries," *Journal of Algorithms*, vol. 41, pp. 69–85, 2001.

[87] J. C. Hart, W. O. Cochran, P. J. Flynn, "Similarity hashing: a computer vision solution to the inverse problem of linear fractals," *Fractals*, vol. 5, pp. 39–50, 1997.

[88] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, pp. 13–30, 1963.

[89] J. E. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, pp. 225–231, 1973.

[90] P. Indyk, "Dimensionality reduction techniques for proximity problems," in: *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 371–378, 2000.

[91] P. Indyk, R. Motwani, P. Raghavan and S. Vempala, "Locality-preserving hashing in multidimensional spaces," in: *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pp. 618–625, 1997.

[92] P. Indyk and N. Thaper, "Fast image retrieval via embeddings," in: *Proceedings of the 3rd International Workshop on Statistical and Computational Theories of Vision*, 2003.

[93] S. Janson, "Large deviation inequalities for sums of indicator variables," Technical Report No. 34, Department of Mathematics, Uppsala University, 1994.

[94] S. Janson, "Asymptotic distribution for the cost of linear probing hashing," *Random Structures and Algorithms*, vol. 19 (3–4), pp. 438–471, 2001.

[95] S. Janson, "Individual displacements for linear probing hashing with different insertion policies," Technical Report No. 35, Department of Mathematics, Uppsala University, 2003.

[96] S. Janson, D. E. Knuth, T. Łuczak, and B. Pittel, "The birth of the giant component," *Random Structures and Algorithms*, vol. 4 (3), pp. 233–358, 1993.

[97] S. Janson, T. Luczak, and A. Rucinski, *Random Graphs*, Wiley-Interscience, New York, 2000.

[98] K. Joag-Dev and F. Proschan, "Negative association of random variables, with applications," *Annals of Statistics*, vol. 11 (4), pp. 286–295, 1983.

[99] N. L. Johnson and S. Kotz, *Urn Models and Their Application: An Approach to Modern Discrete Probability Theory*, John Wiley, New York, 1977.

[100] A. Kamath, R. Motwani, K. Palem, and P. Spirakis, "Tail bounds for occupancy and the satisfiability threshold conjecture," *Random Structures and Algorithms*, vol. 7, pp. 59–80, 1995.

[101] R. Karp, M. Luby, and F. Meyer auf der Heide, "Efficient PRAM simulation on a distributed memory machine," *Algorithmica*, vol. 16, pp. 245–281, 1996.

A preliminary version appeared in: *Proceedings of the 24th ACM Symposium on Theory of Computing (STOC)*, pp. 318–326, 1992.

[102] D. E. Knuth, "Notes on "open" addressing," unpublished notes, 1963. Available at http://www.wits.ac.za/helmut/first.ps.

[103] D. E. Knuth, *The Art of Computer Programming, vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

[104] D. E. Knuth, "Linear probing and graphs, average-case analysis for algorithms," *Algorithmica*, vol. 22 (4), pp. 561–568, 1998.

[105] V. F. Kolchin, B. A. Sevast'yanov and V. P. Chistyakov, *Random Allocations*, V. H. Winston & Sons, Washington, D.C., 1978.

[106] A. G. Konheim and B. Weiss, "An occupancy discipline and applications," *SIAM Journal on Applied Mathematics*, vol. 14, pp. 1266–1274, 1966.

[107] S. Kotz and N. Balakrishnan, "Advances in urn models during the past two decades," in: *Advances in Combinatorial Methods and Applications to Probability and Statistics*, Birkhauser, Boston, pp. 203–257, 1997.

[108] P.-Å. Larson, "Linear hashing with partial expansions," in: *Proceedings of the 6th International Conference on Very Large Databases*, pp. 224–232, 1980.

[109] P.-Å. Larson, "Analysis of uniform hashing," *Journal of the ACM*, vol. 30 (4), pp. 805–819, 1983.

[110] P.-Å. Larson, "Dynamic hash tables," *Communications of the ACM*, vol. 31 (4), pp. 446–457, 1988.

[111] E. L. Lehmann, "Some concepts of dependence," *Annals of Mathematical Statistics*, vol. 37, pp. 1137–1153, 1966.

[112] N. Linial and O. Sasson, “Non-expansive hahsing,” in: *Proceedings of the 28th ACM Symposium on Theory of Computing (STOC)*, pp. 509–517, 1996.

[113] W. Litwin, “Linear hashing: A new tool for files and tables addressing,” in: *Proceedings of the 6th International Conference on Very Large Databases*, pp. 212–223, 1980.

[114] M. L. Łuczak and E. Upfal, “Reducing network congestion and blocking probability through balanced allocation,” in: *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 587–595, 1999.

[115] G. S. Lueker and M. Molodowitch, “More analysis of double hashing,” *Combinatorica*, vol. 13 (1), pp. 83–96, 1993.

[116] G. Lugosi, “Concentration-of-measure inequalities,” presented at the *Workshop on Combinatorics, Probability and Algorithms*, Montreal, 2003. Available at http://www.econ.upf.es/~lugosi/anu.ps.

[117] V. C. H. Ma and M. D. McCool, “Low latency photon mapping using block hashing,” in: *SIGGRAPH/Eurographics Graphics Hardware Workshop*, pp. 89–98, 2002. Full version is available as Technical Report CS-2002-15, School of Computer Science, University of Waterloo, 2002.

[118] J. A. T. Madison, “Fast lookup in hash tables with direct rehashing,” *The Computer Journal*, vol. 23 (2), pp. 188–189, 1980.

[119] E. G. Mallach, “Scatter storage techniques: a uniform viewpoint and a method for reducing retrieval times,” *The Computer Journal*, vol. 20 (2), pp. 137–140, 1977.

[120] C. L. Mallows, “An inequality involving multinomial probabilities,” *Biometrika*, vol. 55, pp. 422–424, 1968.

[121] W. D. Maurer and T. G. Lewis, "Hash table methods," *ACM Computing Surveys*, vol. 7 (1), pp. 5–19, 1975.

[122] C. McDiarmid, "On the method of bounded differences," in: *Surveys in Combinatorics 1989*, vol. 141, pp. 148–188, London Mathematical Society Lecture Notes Series, Cambridge University Press, Cambridge, 1989.

[123] C. McDiarmid, "Concentration," in: *Probabilistic methods for algorithmic discrete mathematics*, (M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, eds.), Springer, pp. 195–248, 1998.

[124] K. Mehlhorn and U. Vishkin, "Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memory," *Acta Informatica*, vol. 21, pp. 339–374, 1984.

[125] H. Mendelson and U. Yechiali, "A new approach to the analysis of linear probing schemes," *Journal of the ACM*, vol. 27 (3), pp. 474–483, 1980.

[126] F. Meyer auf der Heide and M. Dietzfelbinger, "High performance universal hashing with applications to shared memory simulations, data structures and efficient algorithms," in: *Data Structures and Efficient Algorithms*, LNCS 594, Springer-Verlag, pp. 250–269, 1992.

[127] F. Meyer auf der Heide, C. Scheideler, and V. Stemann, "Exploiting storage redundancy to speed up randomized shared memory simulations," in: *Theoretical Computer Science*, Series A, vol. 162 (2), pp. 245–281, 1996. Extended abstract appeared in: *Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science*, pp. 267–278, 1995.

[128] F. Meyer auf der Heide, K. Schröder, and F. Schwarze, "Routing on networks of optical crossbars," in: *Theoretical Computer Science*, 196, pp. 181–200, 1998.

[129] P. B. Miltersen, "Error correcting codes, perfect hashing circuits, and deterministic dynamic dictionaries," in: *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 556–563, 1998.

[130] M. D. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*, Ph.D. thesis, Computer Science Department, University of California at Berkeley, 1996.

[131] M. D. Mitzenmacher, "Load balancing and density dependent jump Markov processes," in: *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 213–222, 1996.

[132] M. D. Mitzenmacher, "Studying balanced allocations with differential equations," *Combinatorics, Probability, and Computing*, vol. 8, pp. 473–482, 1999. Full version available as Technical Note 1997–024, Digital Equipment Corp. Systems Research Center.

[133] M. D. Mitzenmacher, B. Prabhakar, and D. Shah, "Balls and bins with memory," in: *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 799–808, 2002.

[134] M. D. Mitzenmacher, A. Richa, and R. Sitaraman, "The power of two random choices: A survey of the techniques and results," in: *Handbook of Randomized Computing*, (P. Pardalos, S. Rajasekaran, and J. Rolim, eds.), pp. 255–305, 2000.

[135] M. D. Mitzenmacher and B. Vöcking, "The asymptotics of selecting the shortest of two, improved," in: *Proceedings of the 37th Allerton Conference on Communication, Control, and Computing*, pp. 326–327, 1998. Full version available as

Technical Report TR–08–99, Department of Computer Science, Harvard University, Cambridge, MA, 1999.

[136] R. Morris, "Scatter storage techniques," *Communications of the ACM*, vol. 11 (1), pp. 38–44, 1968.

[137] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, MA, 1995.

[138] J. I. Munro and P. Celis, "Techniques for collision resolution in hash tables with open addressing," in: *Proceedings of 1986 Fall Joint Computer Conference*, pp. 601–610, 1999.

[139] M. Okamoto, "Some inequalities relating to the partial sum of binomial probabilitie," *Annals of Mathematical Statistics*, vol. 10, pp. 29–35, 1958.

[140] A. Östlin and R. Pagh, "Simulating uniform hashing in constant time and optimal space," Technical Report RS-02-27, BRICS, Department of Computer Science, University of Aarhus, 2002.

[141] A. Östlin and R. Pagh, "Uniform hashing in constant time and linear space," in: *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pp. 622-628, 2003.

[142] R. Pagh, "Hash and displace: Efficient evaluation of minimal perfect hash functions," in: *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, LNCS 1663, Springer-Verlag, pp. 49–54, 1999.

[143] R. Pagh, "A trade-off for worst-case efficient dictionaries," *Nordic Journal of Computing*, vol. 7, pp. 151–163, 2000. A preliminary version appeared in: *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, LNCS 1851, Springer-Verlag, pp. 22-31, 2000.

[144] R. Pagh, "On the cell probe complexity of membership and perfect hashing," in: *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pp. 425–432, 2001.

[145] R. Pagh, *Hashing, Randomness and Dictionaries*, Ph.D. thesis, University of Aarhus, 2002.

[146] R. Pagh and F. F. Rodler, "Cuckoo hashing," in: *Proceedings of the European Symposium on Algorithms*, LNCS 2161, Springer-Verlag, pp. 121–133, 2001. A previous version is available as BRICS Report Series RS–01–32, Department of Computer Science, University of Aarhus, 2001.

[147] W. W. Peterson, "Addressing for random-access storage," *IBM Journal of Research and Development*, vol. 1 (2), pp. 130–146, 1957.

[148] G. C. Pflug and H. W. Kessler, "Linear probing with a nonuniform address distribution," *Journal of the ACM*, vol. 34 (2), pp. 397–410, 1987.

[149] B. Pittel, "Linear probing: The probable largest search time grows logarithmically with the number of records," *Journal of Algorithms*, vol. 8, pp. 236–249, 1987.

[150] B. Pittel, J. Spencer, and N. Wormald, "Sudden Emergence of a Giant k-Core in a Random Graph," *Journal of Combinatorial Theory*, Series B, vol. 67, pp. 111–151, 1996.

[151] P. V. Poblete and J. I. Munro, "Last-Come-First-Served hashing," *Journal of Algorithms*, vol. 10, pp. 228–248, 1989.

[152] P. V. Poblete, A. Viola, and J. I. Munro, "Analyzing the LCFS linear probing hashing algorithm with the help of Maple," *Maple Technical Newletter*, vol. 4 (1), pp. 8–13, 1997.

[153] M. Raab and A. Steger, "“Balls into bins”—a simple and tight analysis," in: *Proceedings of the 2nd Workshop on Randomization and Approximation Techniques in Computer Science*, vol. 1518, Lecture Notes in Computer Science, Springer-Verlag, pp. 159–170, 1998.

[154] R. L. Rivest, "Optimal arrangement of keys in a hash table," *Journal of the ACM*, vol. 25 (2), pp. 200–209, 1978.

[155] J. T. Robinson, "Order preserving linear hashing using dynamic key statistics," in: *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp. 91–99, 1986.

[156] W. Rudin, *Real and Complex Analysis*, McGraw-Hill, New York, 1987.

[157] T. Schickinger, A. Steger, "Simplified witness tree arguments," in: *Proceedings of the 27th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2000)*, LNCS 1963, Springer-Verlag, pp. 71–87, 2000.

[158] J. P. Schmidt and A. Siegel, "Double hashing is computable and randomizable with universal hash functions," submitted. A full version is available as Technical Report TR1995-686, Computer Science Department, New York University, 1995.

[159] D. Shah, B. Prabhakar, "The use of memory in randomized load balancing," in: *Proceedings of the IEEE International Symposium on Information Theory*, 2002.

[160] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter sensitive hashing," in: *Proceedings of the 9th IEEE International Conference on Computer Vision*, vol. 2, pp. 750–760, 2003.

[161] A. Siegel, "On universal classes of extremely random constant time hash functions and their time-space tradeoff," Technical Report TR1995-684, Computer Science Department, New York University, 1995. A previous version appeared under the title "On universal classes of fast high performance hash functions, their time-space tradeoff and their applications," in: *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 20–25, 1989.

[162] A. Siegel and J. P. Schmidt, "Closed hashing is computable and optimally randomizable with universal hash functions," submitted. A full version is available as Technical Report TR1995-687, Computer Science Department, New York University, 1995.

[163] V. Stemann, "Parallel balanced allocations," in: *Proceedings of the 8th ACM Symposium on Parallel Algorithms and Architectures*, pp. 261–269, 1996.

[164] J. D. Ullman, "A note on the efficiency of hashing functions," *Journal of the ACM*, vol. 19 (3), pp. 569–575, 1972.

[165] A. Viola, "Exact distributions of individual displacements in linear probing hashing." Preprint, 2003.

[166] A. Viola and P. V. Poblete, "The analysis of linear probing hashing with buckets," *Algorithmica*, vol. 21, pp. 37–71, 1998.

[167] J. S. Vitter, *Analysis of Coalescing Hashing*, Ph.D. thesis, Department of Computer Science, Stanford University, 1980.

[168] J. S. Vitter W.-C. Chen, *Design and Analysis of Coalesced Hashing*, Oxford University Press, New York, 1987.

[169] J. S. Vitter and P. Flajolet, "Average-case analysis of algorithms and data structures," in: *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, ed. J. van Leeuwen, pp. 431–524, MIT Press, Amsterdam, 1990.

[170] B. Vöcking, "How asymmetry helps load balancing," in: *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 131–141, 1999.

[171] B. Vöcking, "Symmetric vs. asymmetric multiple-choice algorithms," in: *Proceedings of the 2nd ARACNE Workshop*, Aarhus, pp. 7–15, 2001.

[172] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich, "Queueing system with selection of the shortest of two queues: An asymptotic approach," *Problems of Information Transmission*, vol. 32 (1), pp. 15–27, 1996.

[173] M. N. Wegman and J. L. Carter, "New classes and applications of hash functions," in: *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 175–182, 1979.

[174] M. N. Wegman and J. L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, vol. 22 (3), pp. 265–279, 1981.

[175] R. L. Wheeden and A. Zygmund, *Measure and Integral*, Marcel Dekker, New York, 1977.

[176] P. Woelfel, "Efficient strongly universal and optimally universal hashing," in: *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, LNCS 1672, Springer-Verlag, pp. 262–272, 1999.

[177] J. Wu and L. Kobbelt, “Fast mesh decimation by multiple-choice techniques,” in: *Proceedings of Vision, Modeling, and Visualization*, pp. 241–248, 2002.

[178] A. C. Yao, “Uniform hashing is optimal,” *Journal of the ACM*, vol. 32 (3), pp. 687–693, 1985.