On the k-Independence Required by Linear Probing and Minwise Independence

Mihai Pătraşcu and Mikkel Thorup

AT&T Labs

Abstract. We show that linear probing requires 5-independent hash functions for expected constant-time performance, matching an upper bound of [Pagh et al. STOC'07]. For $(1+\varepsilon)$-approximate minwise independence, we show that $\Omega(\lg \frac{1}{\varepsilon})$-independent hash functions are required, matching an upper bound of [Indyk, SODA'99]. We also show that the multiply-shift scheme of Dietzfelbinger, most commonly used in practice, fails badly in both applications.

1 Introduction

The concept of k-wise independence was introduced by Wegman and Carter [20] in FOCS'79 and has been the cornerstone of our understanding of hash functions ever since. Formally, a family $\mathcal{H} = \{h : [u] \to [b]\}$ of hash functions is k-independent if (1) for any distinct keys $x_1, \ldots, x_k \in [u]$, the hash codes $h(x_1), \ldots, h(x_k)$ are independent random variables; and (2) for any fixed x, $h(x)$ is uniformly distributed in $[b]$.

As the concept of independence is fundamental to probabilistic analysis, k-independent functions are both natural and powerful in algorithm analysis. They allow us to replace the heuristic assumption of truly random hash functions with real (implementable) hash functions that are still "independent enough" to yield provable performance guarantees. We are then left with the natural goal of understanding the independence required by algorithms.

The canonical construction of a k-independent family is based on polynomials of degree $k-1$. Let $p \geq u$ be prime. Picking random $a_0, \ldots, a_{k-1} \in \{0, \ldots, p-1\}$, the hash function is defined by:

$$h(x) = \Big(\big(a_{k-1} x^{k-1} + \cdots + a_1 x + a_0 \big) \bmod p \Big) \bmod b$$

For $p \gg b$, the hash function is statistically close to k-independent.

In simple cases, 2-independence suffices. For instance, if one implements a hash table by chaining, the time it takes to query x is proportional to the number of keys y colliding with x (i.e. $h(x) = h(y)$). Thus, pairwise independence of $h(x)$ and $h(y)$ is all we need to understand the expected query time.

At the other end of the spectrum, $O(\lg n)$-independence suffices in a vast majority of applications. One reason for this is the Chernoff bounds of [15] for k-independent events, whose probability bounds differ from the full-independence

Chernoff bound by $2^{-\Omega(k)}$. Another reason is that random graphs with $O(\lg n)$-independent edges [1] share many of the properties of truly random graphs.

In this paper, we study two compelling applications in which independence $2 < k < \lg n$ is currently needed: linear probing and minwise-independent hashing. (The reader unfamiliar with these applications will find more details below.) For linear probing, Pagh et al. [11] showed that 5-independence suffices, thus giving the first realistic implementation of linear probing with formal guarantees. For minwise-independence, Indyk [9] showed that ε approximation can be obtained using $O(\lg \frac{1}{\varepsilon})$-independence.

In both cases, it was known that 2-independence does not suffice [11, 3], and, indeed, the simplest family $x \mapsto (ax + b) \bmod p$ provides a counterexample. However, a significant gap remained to the upper bounds.

In this paper, we close this gap, showing that both upper bounds are, in fact, tight. We do this by exhibiting carefully constructed families for which these algorithms fail: for linear probing, we give a 4-independent family that leads to $\Omega(\lg n)$ query time; and for minwise independence, we give an $\Omega(\lg \frac{1}{\varepsilon})$-independent family that leads to 2ε approximation.

Concrete schemes. Our results give a powerful understanding of a natural combinatorial resource (independence) for two important algorithmic questions. In other words, they are limits on how far the *paradigm* of independence can bring us. Note, however, that independence is only one property that concrete hash schemes have. In a particular application, a hash scheme can behave much better that its independence guarantees, if it has some other probabilistic property unrelated to independence.

In practice, the most popular hash function is not $x \mapsto \big((ax+b) \bmod p\big) \bmod u$, but Dietzfelbinger's multiply-shift scheme [6], which can be twice as fast [17]. To hash w-bit integers to the range $b = 2^\ell$, the scheme picks a random a of $2w$ bits, and computes $(ax) >> (2w - \ell)$, where $>>$ denotes unsigned shift.

In Appendix B, we prove that linear probing with multiply-shift hashing suffers from $\Omega(\lg n)$ expected running times. Similarly, minwise independent hashing has a very large approximation, of $\varepsilon = \Omega(\lg n)$. While these results are not surprising, given the "moral similarity" of multiply-shift and $ax+b$ schemes, they do require rather involved arguments. We feel this effort is justified, as it brings the theoretical lower bounds in line with programming reality.

In the same vein, one could ask to replace our lower bounds, which construct artificial families of high independence, by a proof that the family of polynomials fails. While this is an intriguing algebraic question, we do note that it is far less pressing from a practical viewpoint. Even with the best known implementation tricks (such as computing modulo Mersenne primes), higher degree polynomials make rather slow hash functions, and are not widely used. For instance, the fastest known 5-independent family is tabulation based [18, 19], and it outperforms polynomials by a factor of 5 or more. In a recent manuscript, we show [13] that tabulation-based hash functions do, in fact, behave better than their independence would suggest: for instance, 3-independent tabulation yields $O(1)$ running times for linear probing.

1.1 Technical Discussion: Linear Probing

Linear probing uses a hash function to map a set of keys into an array of size b. When inserting x, if the desired location $h(x)$ is already occupied, the algorithm scans $h(x)+1, h(x)+2, \ldots$ until an empty location is found, and places x there. The query algorithm starts at $h(x)$ and scans either until it finds x, or runs into an empty position, which certifies that x is not in the hash table. We assume constant load of the hash table, e.g. the number of keys is $n \leq \frac{2}{3}b$.

This classic data structure is the most popular implementation of hash tables, due to its unmatched simplicity and efficiency. On modern architectures, access to memory is done in cache lines (of much more than one element), so inspecting a few consecutive values typically translates into just one memory probe. Even if the scan straddles a cache line, the behavior will still be better than a second random memory access on architectures with prefetching. Empirical evaluations [2, 8, 12] confirm the practical advantage of linear probing over other known schemes, while cautioning [8, 19] that it behaves quite unreliably with weak hash functions (such as 2-independent). Taken together, these findings form a strong motivation for theoretical analysis.

Linear probing was first shown to take expected constant time per operation in 1963 by Knuth [10], in a report now considered the birth of algorithm analysis. However, this required truly random hash functions.

A central open question of Wegman and Carter [20] was how linear probing behaves with k-independence. Siegel and Schmidt [14, 16] showed that $O(\lg n)$-independence suffices. Recently, Pagh et al. [11] showed that even 5-independent hashing works. We now close this line of work, showing that 4-independence is not enough.

Review of the 5-independence upper bound. To better situate our lower bounds, we begin by reviewing the upper bound of [11]. The main probabilistic tool featuring in this analysis is a 4^{th} moment bound. Consider throwing n balls into b bins uniformly. Let X_i be the probability that ball i lands in the first bin, and $X = \sum_{i=1}^{n} X_i$ the number of balls in the first bin. We have $\mu = \mathbf{E}[X] = \frac{n}{b}$. Then, the k^{th} moment of X is defined as $\mathbf{E}[(X-\mu)^k]$.

As long as our placement of the balls is k-independent, the k^{th} moment is identical to the case of full independence. For instance, the 4^{th} moment is:

$$\mathbf{E}[(X-\mu)^4] = \mathbf{E}\big[\big(\sum_i (X_i - \tfrac{1}{b})\big)^4\big] = \sum_{i,j,k,l} \mathbf{E}\big[(X_i - \tfrac{1}{b})(X_j - \tfrac{1}{b})(X_k - \tfrac{1}{b})(X_l - \tfrac{1}{b})\big].$$

The only question in calculating this quantity is the independence of sets of at most 4 items. Thus, 4-independence preserves the 4^{th} moment of full randomness.

Moments are a standard approach for bounding the probability of large deviations. Let's say that we expect μ items in the bin, but have capacity 2μ; what is the probability of overflow? A direct calculation shows that the 4^{th} moment is $\mathbf{E}[(X-\mu)^4] = O(\mu^2)$. Then, by a Markov bound, the probability of overflow is $\Pr[X \geq 2\mu] = \Pr[(X-\mu)^4 \geq \mu^4] = O(1/\mu^2)$. By contrast, if we only

have 2-independence, we can use the 2^{nd} moment $\mathbf{E}[(X-\mu)^2] = O(\mu)$ and obtain $\Pr[X \geq 2\mu] = O(1/\mu)$. Observe that the 3^{rd} moment is not useful for this approach, since $(X-\mu)^3$ can be negative, so Markov does not apply.

To apply moments to linear probing, we consider a perfect binary tree spanning the array. For notational convenience, let us assume that the load is at most $n \leq b/3$. A node on level ℓ has 2^ℓ array positions under it, and we expect $2^\ell/3$ keys to be hashed to one of them (but more or less keys may actually appear in the subtree, since items are not always placed at their hash position). Call the node dangerous if at least $\frac{2}{3}2^\ell$ keys hash to it.

In the first stage, we will bound the total time it takes to construct the hash table (the cost of inserting n distinct items). If the table consists of runs of $k_1, k_2, \ldots$ elements ($\sum k_i = n$), the cost of constructing it is bounded from above by $O(k_1^2 + k_2^2 + \ldots)$. To bound these runs, we make the following crucial observation: if a run contains between 2^ℓ and $2^{\ell+1}$ elements, then some node at level $\ell - 2$ above it is dangerous.

For a proof, assume the run goes from positions i to j. The interval $[i, j]$ is spanned by 4 to 9 nodes on level $\ell - 2$. Assume for contradiction that none are dangerous. The first node, which is not completely contained in the interval, contributes less than $\frac{2}{3}2^{\ell-2}$ elements to the run (it the most extreme case, this many elements hashed to the last location of that node). But the subsequent nodes all have more than $2^{\ell-2}/3$ free locations in their subtree, so 2 more nodes absorb all excess elements. Thus, the run cannot go on for 4 nodes, contradiction.

This observation gives an upper bound on the cost: add $O(2^{2\ell})$ for each dangerous node at some level ℓ. Denoting by $p(\ell)$ the probability that a node on level ℓ is dangerous, the expected cost is thus $\sum_\ell (n/2^\ell) \cdot p(\ell) \cdot 2^{2\ell} = \sum_\ell n \cdot 2^\ell p(\ell)$. Using the 2^{nd} moment to bound $p(\ell)$, one would obtain $p(\ell) = O(2^{-\ell})$, so the total cost would be $O(n \lg n)$. However, the 4^{th} moment gives $p(\ell) = O(2^{-2\ell})$, so the cost at level ℓ is now $O(n/2^\ell)$. In other words, the series starts to decay geometrically and is bounded by $O(n)$.

To bound the running time of one particular operation (query or insert q), we actually use the stronger guarantee of 5-independence. If the query lands at a uniform place conditioned on everything else in the table, then at each level it will pay the "average cost" of $O(1/2^\ell)$, which sums up to $O(1)$.

Our results. Two intriguing questions pop out of this analysis. First, is the independence of the query really crucial? Perhaps one could argue that the query behaves like an average operation, even if it is not completely independent of everything else. Secondly, one has to wonder whether 3-independence suffices (by using something other than 3^{rd} moment): all that is needed is a bound slightly stronger than 2^{nd} moment in order to make the series decay geometrically!

We answer both questions in strong negative terms. The complete understanding of linear probing with low independence is summarized in Table 1. Addressing the first question, we show that 4-independence cannot give expected time per operation better than $\Omega(\lg n)$, even though n operations take $O(n)$. Our proof demonstrates an important phenomenon: even though most bins have low

Independence	2	3	4	≥ 5
Query time	$\tilde{\Theta}(\sqrt{n})$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
Construction time	$\Theta(n \lg n)$	$\Theta(n \lg n)$	$\Theta(n)$	$\Theta(n)$

Table 1. Expected time bounds with a bad family of k-independent hash functions. Construction time refers to the total time over n different insertions.

load, a particular element's hash code could be correlated with the (uniformly random) choice of *which* bins have high load.

An even more striking illustration of this fact happens for 2-independence: the query time blows up to $\Omega(\sqrt{n})$ in expectation, since we are left with no independence at all after conditioning on the query's hash. This demonstrates a very large separation between linear probing and collision chaining, which enjoys $O(1)$ query times even for 2-independent hash functions.

Addressing the second question, we show that 3-independence is not enough to guarantee even a construction time of $O(n)$. Thus, in some sense, the 4^{th} moment analysis is the best one can hope for.

1.2 Technical Discussion: Minwise Independence

This concept was introduced by two classic algorithms: detecting near-duplicate documents [3, 4] and approximating the size of the transitive closure [5]. The basic step in these algorithms is estimating the size of the intersection of pairs of sets, relative to their union: for A and B, we want to find $\frac{|A \cap B|}{|A \cup B|}$ (the *Jaccard similarity coefficient*). To do this efficiently, one can choose a hash function h and maintain $\min h(A)$ as the sketch of an entire set A. If the hash function is truly random, we have $\Pr[\min h(A) = \min h(B)] = \frac{|A \cap B|}{|A \cup B|}$. Thus, by repeating with several hash functions, or by keeping the bottom k elements with one hash function, the Jaccard coefficient can be estimated up to a small approximation.

To make this idea work, the property that is required of the hash function is *minwise independence*. Formally, a family of functions $\mathcal{H} = \{h : [u] \to [u]\}$ is said to be minwise independent if, for any set $S \subset [u]$ and any $x \notin S$, we have $\Pr_{h \in \mathcal{H}}[h(x) < \min h(S)] = \frac{1}{|S|+1}$. In other words, x is the minimum of $S \cup \{x\}$ only with its "fair" probability $\frac{1}{|S|+1}$.

As good implementations of exact minwise independent functions are not known, the definition is relaxed to ε-minwise independent, where $\Pr_{h \in \mathcal{H}}[h(x) < \min h(S)] = \frac{1 \pm \varepsilon}{|S|+1}$. Using such a function, we will have $\Pr[\min h(A) = \min h(B)] = (1 \pm \varepsilon)\frac{|A \cap B|}{|A \cup B|}$. Thus, the ε parameter of the minwise family dictates the best approximation achievable in the algorithms (which *cannot* be improved by repetition).

Indyk [9] gave the only implementation of minwise independence with provable guarantees, showing that $O(\lg \frac{1}{\varepsilon})$-independent functions are ε-minwise independent.

His proof uses another tool enabled by k-independence: the inclusion-exclusion principle. Say we want to bound the probability that at least one of n events is "good." We can define $p(k) = \sum_{S \subset [n], |S|=k} \Pr[\text{all } S \text{ are good}]$. Then, the probability that at least one event is good is, by inclusion-exclusion, $p(1) - p(2) + p(3) - p(4) + \ldots$. If we only have k-independence (k odd), we can upper bound the series by $p(1) - p(2) + \cdots + O(p(k))$. In the common scenario that $p(k)$ decays exponentially with k, the trimmed series will only differ from the full independence case by $2^{-\Omega(k)}$. Thus, k-independence achieves bounds exponentially close to full independence, whenever probabilities can be computed by inclusion-exclusion. This turns out to be the case for minwise independence: we can express the probability that at least some element in S is below x by inclusion-exclusion.

In this paper, we show that, for any $\varepsilon > 0$, there exist $\Omega(\lg \frac{1}{\varepsilon})$-independent hash functions that are no better than (2ε)-minwise independence. Thus, ε-minwise independence requires $\Omega(\lg \frac{1}{\varepsilon})$ independence.

2 Time $\Theta(\sqrt{n})$ with 2-Independence

We define a 2-independent hash family such that the expected query time is $\Theta(\sqrt{n})$. The main idea of the proof is that the query can play a special role: even if most portions of the hash table are lightly loaded, the query can be correlated with the portions that *are* loaded. We assume that b is a power of two, and we store $n = b/2$ keys. We also assume that $\sqrt{n}$ is a power of two.

We can think of the stored keys and the query key as fixed, and we want to find bad ways of distributing them k-independently into the range $[b]$. To extend the hash function to the entire universe, all other keys are hashed totally randomly. We consider unsuccessful searches, i.e. the search key q is not stored in the hash table. The query time for q is the number of cells considered from $h(q)$ up to the first empty cell. If, for some d, the interval $Q = (h(q) - d, h(q)]$ has $2d$ keys, then the search time is $\Omega(d)$.

Let $d = 2\sqrt{n}$; this is a power of two dividing b. In our construction, we first pick the hash $h(q)$ uniformly. We then divide the range into $\sqrt{n}$ intervals of length d, of the form $(h(q) + i \cdot d, h(q) + (i+1)d]$, wrapping around modulo b. One of these intervals is exactly Q.

We prescribe the distribution of keys between the intervals; the distribution within each interval will be fully random. To place $2d = 4\sqrt{n}$ keys in the query interval with constant probability, we mix among two strategies with constant probabilities (to be determined):

S1: Spread keys evenly, with $\sqrt{n}$ keys in each interval.

S2: Pick 4 intervals including the query interval, and gather all $4\sqrt{n}$ keys in a random one of these. All other intervals get $\sqrt{n}$ keys. With probability $1/4$, it is the query interval that gets overfilled, and the search time is $\Omega(\sqrt{n})$.

To prove that the hash function is 2-independent, we need to consider pairs of two stored keys, and pairs involving the query and one stored key. In either case,

we can just look at the distribution into intervals, since the position within an interval is truly random. Furthermore, we only need to understand the probability of the two keys landing in the same interval (which we call a "collision"). By the above process, if two keys do not collide, they will actually be in uniformly random distinct intervals.

Since store keys are symmetric, the probability of q and x colliding is given by the expected number of items in Q, which is exactly $d/2$. Thus $h(q)$ and $h(x)$ are independent. To analyze pairs of the form (x, y), we will compute the expected number of collisions among stored keys. This will turn out to be $\binom{n}{2}/\sqrt{n}$, proving that x and y collide with probability $1/\sqrt{n}$, and thus, are independent.

In strategy S1, we get the smallest possible number of collisions: $\sqrt{n}\binom{\sqrt{n}}{2} = \frac{1}{2}n^{1.5} - \frac{1}{2}n$. Compared to $\binom{n}{2}/\sqrt{n} = \frac{1}{2}n^{1.5} - \frac{1}{2}\sqrt{n}$, this is too few by almost $n/2$. In S2, we get $(\sqrt{n}-4)\binom{\sqrt{n}}{2} + \binom{4\sqrt{n}}{2} = \frac{1}{2}n^{1.5} + \frac{11}{2}n$, which is too large by almost $5.5n$. To get the right expected number of collisions, we use S2 with probability $\frac{5.5n+o(n)}{(0.5+5.5)n+o(n)} = \frac{11}{12} \pm o(1)$.

It is not hard to prove an upper bound of $O(\sqrt{n})$ on the expected query cost: a higher query time implies too many collisions for 2-independence. Formally, divide into $\sqrt{n}$ intervals of length $d = 2\sqrt{n}$. If the query cost is td, it must pass at least $(t-2)d$ keys in intervals filled with at least d keys. Also, if we have k keys in such full intervals, the number of collisions is $\Omega(k\sqrt{n})$ above the minimum possible. Thus, if the expected query cost is $\omega(d)$, the the expected number of extra collisions is $\omega(n)$, contradicting 2-universality.

3 Construction Time $\Omega(n \lg n)$ with 3-Independence

We will now construct a 3-independent family of hash functions, such that the time to insert n items into a hash table is $\Omega(n \lg n)$. As before, we assume the array size b is a power of two; also, n is a power of two exceeding $\frac{2}{3}b$. Since we are looking at the total cost of n insertions, if some interval of length d gets $d+s$ keys (overflow of s), then these $d+s$ insertions cost $\Omega(s^2)$. We will add up such squared overflow costs over disjoint intervals, and demonstrate a total cost of $\Omega(n \lg n)$. Our proof and notations are in a more general form than needed, laying the ground for our work on 4-independence.

We imagine a perfect binary tree spanning the array. Our hash function will recursively distribute keys from a node to its two children, starting at the root. Nodes run independent random distribution processes. Then, if each node makes a k-independent distribution, overall the function is k-independent.

For a node, will mix between two strategies:

S1: Distribute the keys evenly between the two children (always possible, as n is a power of 2).
S2: Give all the keys to one of the children.

Our first goal is to determine the correct probability for the second strategy, p^{S2}, such that the distribution process is 3-independent. Then we will calculate the cost it induces on linear probing.

To prove 3-independence, we will reason about moments. Our randomized procedure treats keys symmetrically, and ignores the distinction between left/right children. Say the current node has to distribute m keys to its two children, u and v. We want to look at the moments of the number of elements given to u. Formally, let X_a be the indicator random variable for key a ending in u; by symmetry, $\mathbf{E}[X_a] = \frac{1}{2}$. Let $X = \sum_a X_a$ be the number of keys assigned to u. Then $\mu = \mathbf{E}[X] = m/2$ and the k^{th} moment is $F_k = \mathbf{E}[(X-\mu)^k]$. We have the following characterization of 3-independent distributions:

Lemma 1. *The distribution is 3-independent iff $\mu = m/2$ and $F_2 = m/4$.*

Proof. We first observe that, when distributing items into two bins, any odd moment is necessarily zero, as $\Pr[X = \mu + \delta] = \Pr[X = \mu - \delta]$ for any δ. If the distribution is 2-independent or more, the 2^{nd} moment is $F_2 = \sum_a \mathbf{E}[(X_a - \frac{1}{2})^2] = \frac{m}{4}$. This shows the "only if" part.

Now consider arbitrary distinct keys a, b, c. Since our process treats keys symmetrically, to show it is 3-independent we only need to show that $\Pr[X_a \wedge X_b \wedge X_c] = \frac{1}{8}$ and $\Pr[X_a \wedge X_b \wedge \neg X_c] = \frac{1}{8}$. Note, furthermore, that

$$Pr[X_a \wedge X_b] = \Pr[X_a \wedge X_b \wedge X_c] + \Pr[X_a \wedge X_b \wedge \neg X_c]$$

Thus, it is equivalent to show the pair of conditions: $\Pr[X_a \wedge X_b \wedge X_c] = \frac{1}{8}$ and $\Pr[X_a \wedge X_b] = \frac{1}{4}$. For notational convenience, let $p_k = \Pr[X_1 \wedge \cdots \wedge X_k]$; this is defined for any distinct keys $X_1, \ldots, X_k$, since keys are symmetric. Our goal is thus to show that p_2 and p_3 have the correct values.

We will show a stronger statement: in general, $p_2, \ldots, p_k$ are determined by $F_2, \ldots, F_k$. In this case, any distribution that has the same moments as a 3-independent distribution must have the same p_2 and p_3 as a 3-independent distribution, i.e. it must be 3-independent!

Let $m^{\bar{k}} = m(m-1)\ldots(m-k+1)$ be the falling factorial. Our precise claim:

Fact 1 *For any k, $m^{\bar{k}} p_k = F_k + f_k(m, F_2, .., F_{k-1})$, for some function f_k.*

For the proof, note that $F_k = \mathbf{E}[(X-\mu)^k] = \mathbf{E}[X^k] + f\big(\mu, \mathbf{E}[X^2], \ldots, \mathbf{E}[X^{k-1}]\big)$. Thus, the goal is to write all $\mathbf{E}[X^j]$, $j = 2$ to k, as a function of $p_2, \ldots, p_k$. For this, note that $\mathbf{E}[X^k] = m^{\bar{k}} p_k + f_1(m,k) p_{k-1} + f_2(m,k) p_{k-2} + \ldots$ □

With this characterization, we can decide on the right mix of S1 and S2 to make the process 3-independent. In strategy S1 (balanced distribution), $X = \mu \pm 1$, so $F_2^{S1} \leq 1$. In S2 (all to one child), $|X - \mu| = m/2$ so $F_2^{S2} = m^2/4$. Hence, the proper balancing is $p^{S2} = \frac{1}{m} \pm O(\frac{1}{m^2})$, yielding the desired 2^{nd} moment $m/4$. By Lemma 1, this distribution is 3-independent.

We now calculate the cost in terms of squared overflows. As long as the recursive steps spread the keys evenly, the load stays unchanged above 2/3. However, a first time we collect m keys into one child, that interval of the array will get a load of 4/3. This is an overflow of $\Omega(m)$ keys, thus a cost of $\Omega(m^2)$. Since $p^{S2} = \Theta(1/m)$, the expected cost induced by the node is $\Theta(m)$.

However, to avoid double charging, we may only consider the node if there has been not collection in one of his ancestors. As long as S1 applies, the number of keys at depth i is $m \approx n/2^i$, so the probability of the collection strategy is $p^{S2} = \Theta(1/m) = \Theta(2^i/n)$. The probability that a node at depth i is still relevant (no collection among his ancestors) is at least $1 - \sum_{j=0}^{i-1} \Theta(2^j/n) = 1 - \Theta(2^i/n) \geq \frac{1}{2}$ for $i \ll \lg n$. We conclude that the expected cost of each node is linear in the size of its subtree. Hence, the total expected cost is $\Omega(n \lg n)$.

4 Expected Query Time $\Omega(\lg n)$ with 4-Independence

The proof is a combination of the ideas for 2-independence and 3-independence, plus the (severe) complications that arise. As for 2-independence, we will first choose $h(q)$ and then make the stored keys cluster preferentially around $h(q)$. This clustering will actually be the 3-independent type of clustering from before, but done only on the path from the root to $h(q)$. Since this is used for so few nodes, we can balance the 4^{th} moment to give 4-independence, by doing a slightly skewed distribution in the nodes off the query path. Full details appears in Appendix A.

5 Minwise Independence via k-Independence

We will show that it is limited how good minwise independence we can assume based on k-independent hashing. For a given k, our goal is to construct a k-independent scheme over n regular keys and a query key q such that the probability that q gets the minimal hash value is too large by a factor of $1 + 2^{-\Omega(k)}$.

The construction of the distribution is as follows. We assume that k divides n and that k is even. The hash values are going to be uniformly distributed in the unit interval $[0, 1)$. For our construction we divide the unit interval into subintervals of length k/n, that is subintervals of the form $[ik/n, (i+1)k/n)$ for $i = 0, .., n/k - 1$.

The regular keys are distributed totally randomly between the above subintervals. Each subinterval I gets an expected number of k regular keys. We say that I is "exact" if it gets exactly k regular keys. If I is not exact, the regular keys are placed totally randomly within I.

For exact intervals I, we introduce a "parity distribution" based on a parity parameter $P = \{0, 1\}$. The distribution of the k regular keys in I is random subject to the constraint that P determines the parity of the number of keys landing in the first half of I. This means that any $k - 1$ keys are totally independent, but then P determines whether or not the last key should go in the first half. It follows that the overall distribution of the regular keys is $(k-1)$-independent regardless of the parities. It also follows that the overall distribution is k-independent if each exact interval has parity P with probability $1/2$. This does not require independendence of parities for different exact intervals.

The query is generated independently, landing in some subinterval, called the "query interval". The distribution of parities will be determined by this

event. If a given exact interval I gets the query key, its parity is set to even ($P_I = 0$). The probability of this event is k/n. If I does not get the query, its parity is only set to even with probability $(1/2 - k/n)/(1 - k/n)$. It follows that the overall probability that the parity of I is even is exactly $1/2$. Since this is the case for every exact interval, we conclude that the distribution of regular keys is k-independent. Moreover, since the distribution of the regular keys is $(k-1)$-independent regardless of the parities, hence regardless of the query key, we conclude that the distribution of all keys is k-independent, as required.

To understand the effect of the parity, consider an exact interval I with k regular keys plus the query key. We are interested in the "min-probability" that the query becomes smallest in I. With the parity P as a parameter, we denote the min-probability $p^P_{\min}$. With a totally random distribution, the min-probability is the average of $p^0_{\min}$ and $p^1_{\min}$, but in our special distribution the min-probability is $p^0_{\min}$. Our goal is therefore to show that $p^0_{\min}$ is larger than $p^1_{\min}$.

For our analysis, we generate the distribution with parity P as follows. First we distribute $k-1$ keys randomly. Let j be the number of keys in the first half. If j has parity P, we place the last key in the last half. Otherwise, we place it in the first half, which then gets $j' = j + 1$ keys.

For given P, consider some $i > 0$ with parity P. The probability p_i that we get i regular keys in the first half is

$$p_i = \frac{\binom{k-1}{i-1} + \binom{k-1}{i}}{2^{k-1}} = \frac{\binom{k-1}{i-1}(1 + \frac{k-i}{i})}{2^{k-1}} = \frac{\binom{k}{i}}{2^{k-1}}.$$

When the first half has $i > 0$ keys, the min-probability is the probability that the query is in the first half and that it is smaller than the i regular keys in the first half, so the min-probability is $1/(i+1)/2$. The case of i regular keys in the first half therefore contributes

$$r_i = p_i/(i+1)/2 = \binom{k}{i}/(i+1)/2^k,$$

to $p^P_{\min}$. We shall also use this formal definition of r_i when $i = 0$, so $r_0 = 1/2^k$.

If $P = 0$, we also need to consider the case of $i = 0$ regular keys in the first half. The probability of this event is $1/2^{k-1}$. Then the query is smallest if either it ends in the first half, or it ends in the second half as smaller than the k regular keys there. The min-probability when $i = P = 0$ is therefore $1/2 + 1/(k+1)/2$, so this case contributes $1/2^k + 1/(k+1)/2^k = r_0 + 1/(k+1)/2^k$ to $p^0_{\min}$.

Summing up over all cases, we get that $p^0_{\min} = \sum_{i \text{ even}} r_i + 1/(k+1)/2^k$ while it for odd distributions is $p^1_{\min} = \sum_{i \text{ odd}} r_i$.

Now for $i < k$, using the formal definition of r_i, we get that

$$\begin{aligned} r_i &= \binom{k}{i}/(i+1)/2^k = k^{\bar{i}}/i!/(i+1)/2^k \\ &= \binom{k}{i+1}/(k-i)/2^k = \binom{k}{k-i-1}/(k-i)/2^k \\ &= r_{k-i-1}. \end{aligned}$$

The important point here is that if k is even, then $k-i-1$ is odd. Moreover, for $i = 0, 2, .., k-2$, we have $k-i-1 = k-1, k-3, ..., 1$. Therefore we conclude that $\sum_{i\in\{0,2,\ldots,k-2\}} r_i = \sum_{i \text{ odd}} r_i$. This gives us the desired difference between $p^0_{\min}$ and $p^1_{\min}$:

$$p^0_{\min} = p^1_{\min} + r_k + 1/(k+1)/2^k = p^1_{\min} + 2/(k+1)/2^k.$$

With our special distribution, the min-probability is $p^0_{\min}$. With a totally random distribution, the min-probability $p^{rand}_{\min} = 1/(k+1)$ is the average of $p^0_{\min}$ and $p^1_{\min}$. It follows that

$$p^0_{\min} = p^{rand}_{\min} + 1/(k+1)/2^k = p^{rand}_{\min}(1 + 1/2^k).$$

We are now almost done. For the query to be smallest in either distribution, it has to be in the smallest non-empty interval. If this interval is exact, then our even parity increases the chance that the query is smallest by a factor $(1 + 1/2^k)$; otherwise our special distribution does not change the probability. All that remains is to show the exact case happens with probability $\Omega(1/\sqrt{k})$.

The probability that there are no regular keys in $[0, 3(\ln n)/n)$ is $(1 - 3(\ln n)/n)^n < 1/n^3$, so we can ignore this case. We now condition on the query key being in the smallest non-empty subinterval Q. Then all smaller subintervals are empty, so the regular keys are distributed randomly between the intervals from Q and up. If $\mu < 3(\ln n)/n$ is the start of Q, then each of the n regular keys ends in Q with independent probability $k/n/(1-\mu)$. The number landing in Q therefore follows a binomial distribution with mean very close to k, so the probability that the number is exactly k is $\Omega(1/\sqrt{k})$. Thus, overall, we conclude that the probability that the query gets the minimum value is a factor $1 + \Omega(1/(\sqrt{k}2^k))$ larger than it should be.

References

1. Noga Alon and Asaf Nussboim. k-wise independent random graphs. In *Proc. 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 813–822, 2008.
2. John R. Black, Charles U. Martel, and Hongbin Qi. Graph and hashing algorithms for modern architectures: Design and performance. In *Proc. 2nd International Workshop on Algorithm Engineering (WAE)*, pages 37–48, 1998.
3. Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000. See also STOC'98.
4. Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks*, 29:1157–1166, 1997.
5. Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997. See also STOC'94.
6. Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569–580, 1996.

7. Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.
8. Gregory L. Heileman and Wenbin Luo. How caching affects hashing. In *Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX)*, page 141154, 2005.
9. Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001. See also SODA'99.
10. Donald E. Knuth. Notes on open addressing. Unpublished memorandum. See `http://citeseer.ist.psu.edu/knuth63notes.html`, 1963.
11. Anna Pagh, Rasmus Pagh, and Milan Ružić. Linear probing with constant independence. *SIAM Journal on Computing*, 2007. To appear. See also STOC'07.
12. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. See also ESA'01.
13. Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. Manuscript, 2010.
14. Jeanette P. Schmidt and Alan Siegel. The analysis of closed hashing under limited randomness. In *Proc. 22nd ACM Symposium on Theory of Computing (STOC)*, pages 224–234, 1990.
15. Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. See also SODA'93.
16. Alan Siegel and Jeanette P. Schmidt. Closed hashing is computable and optimally randomizable with universal hash functions. Technical Report TR1995-687, Currant Institute, 1995.
17. Mikkel Thorup. Even strongly universal hashing is pretty fast. In *Proc. 11th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–497, 2000.
18. Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 615–624, 2004.
19. Mikkel Thorup and Yin Zhang. Tabulation based 5-universal hashing and linear probing. In *Proc. 12th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2009.
20. Mark N. Wegman and Larry Carter. New classes and applications of hash functions. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. See also FOCS'79.

A Expected Query Time $\Omega(\lg n)$ with 4-Independence

For the first $i_0 = \lceil 2/3 \log_2 n \rceil$ levels, we just do a perfectly random recursive distributions, which in particular is 4-independent, and now all intervals are of length $\Theta(n^{1/3})$. On the subsequent levels, we will apply the 3-independent distribution from the last subsection on the query interval. This will have a too high 4^{th} moment. To balance it, on the non-query intervals, we will apply another 3-independent distribution with a too low 4^{th} moment, arguing that the overall distribution becomes 4-independent. If we reach level $\lceil 0.9 \log_2 n \rceil$ or if at some level, the query interval gets all keys collected on one side, then on all remaining levels, we revert to perfectly random distributions.

As described above, our expected asymptotic cost for the search is $\Omega(\lg n)$ as it was for insertions in the last section. More precisely, after the first i_0, the

intervals still have a fill tightly concentrated around $2/3$, and in particular we can assume this to be the case for the query interval. On level i, $i_0 \leq i < 0.9 \log_2 n$, the probability that the query interval has not yet been collected is $O(2^i/n) = o(1)$. In the non-collected case, we have roughly $n/2^i$ keys that get clustered with probability $\Theta(1/m)$ leading to an overflow and cost of $\Omega(m)$. The expected cost on level i is thus $\Omega(1)$. We have $\Theta(\lg n)$ such levels, so our expected search cost is $\Omega(\lg n)$.

As described above, however the query key is placed, we distributed the stored keys 3-independently. This means that any set of 3 stored keys plus the query key are distributed independently. Hence, for 4-independence, it only remains to prove that any four stored keys are distributed independently. We will achieve this inductively, one level at the time, observing the following condition:

Property 1. For any set K of at most 4 keys, conditioned on them all being in the same interval on level i, they distribute fully randomly on the two subintervals on level $i + 1$.

As stated in the construction, we will ascertain that all recursive calls are 3-independent, implying that Property 1 is satisfied when $|K| \leq 3$.

We now argue that assuming 3-independence, we only have to worry about the probability $P_4 = P_{(\{a,b,c,d\},\emptyset)}$ that any four keys a, b, c, d, all end on the left side. This is because $P_{(\{a,b,c\},\{d\})} = P_{(\{a,b,c\},\emptyset)} - P_{(\{a,b,c,d\},\emptyset)}$, and $P_{(\{a,b\},\{c,d\})} = P_{(\{a,b\},\{c\})} - P_{(\{a,b,d\},\{c\})}$. From 3-independence, we know that $P_{(\{a,b,c\},\emptyset)}$ and $P_{(\{a,b\},\{c\})}$ have the correct value of 2^{-3}, so if $P_{(\{a,b,c,d\},\emptyset)}$ has the correct value of 2^{-4}, then so does $P_{(\{a,b,c\},\{d\})}$ and $P_{(\{a,b\},\{c,d\})}$. Thus

Lemma 2. *If a symmetric distribution on two subintervals is 3-independent and $P_{(4,0)} = 2^{-4}$, then the distribution is 4-independent.*

We will now seek to get the correct probability $P_{(4,0)} = 1/2^4$ of clustering the four keys, conditioned only on them being in a given interval on level i. The intervals on level i are of the form $(q + jt/2^i, q + (j + 1)t/2^i] \pmod t$. For the location of the intervals, we only care about the offset $o = q \bmod t/2^i$. For the distribution we see on level i, we can think of q as a random number conditioned on the offset. Thus, from level i, it is random which interval is the query interval. Moreover, our distribution of stored keys is immune to permutations. On level i, we end up with a certain number of quadruples in the query interval and a certain number of quadruples that are together in other intervals. This means conditioned on the set K being in a certain interval $I = (o+jt/2^i, o+(j+1)t/2^i]$, there is a certain probability p_{query} that I is the query interval. In that case we will get a certain probability $P_{(4,0)}^{\text{query}} > 1/2^4$ that they end in the left interval. Otherwise, there will be a smaller probability $P_{(4,0)}^{\text{non-query}} < 1/2^4$ designed to satisfy that $p_{query} P_{(4,0)}^{\text{query}} + (1 - p_{query}) P_{(4,0)}^{\text{non-query}} = 1/2^4$. It would be tempting to try to do the above balancing directly via the 4^{th} moment, but because the different intervals on level i have different number of keys, there is no simple relation between balancing the 4^{th} moment and balancing $P_{(4,0)}$. Nevertheless, we shall use the 4^{th} moment to control $P_{(4,0)}$.

From Fact 1 we get that $m^{\bar{k}}P_k = F_k + f_{k-1}(m, F_2, .., F_{k-1})$. Let C_{k-1}^{random} be the value of $f_{k-1}(m, F_2, .., F_{k-1})/m^{\bar{k}}$ when the distribution is truly random. Then

$$P_{(k,0)} = F_k/m^{\bar{k}} + C_{k-1}^{\text{random}}$$

for any $k-1$ universal function. We are interested in the case where $k = 4$. To compute C_3^{random}, we derive the truly random values of $P_{(4,0)}^{\text{random}}$ and F_4^{random}. Trivially $P_{(4,0)}^{\text{random}} = 1/2^4$ and

$$F_4^{\text{random}} = m/2^4 + \binom{4}{2} m^{\bar{2}}/2^4 = (6m^2 - 5m)/2^4.$$

Hence

$$C_3^{\text{random}} = F_4^{\text{random}}/m^{\bar{4}} - P_{(4,0)}^{\text{random}} = 1/2^4(1 - (6m^2 - 5m)/m^{\bar{4}}).$$

We will now compute $P_{(4,0)}^{\text{query}}$ for the case of the query interval. Here we collected all keys on one side with probability $p_{\text{collect}} = 1/m - O(1/m^2)$, and then we get a 4^{th} moment of $F_4^{\text{collect}} = (m/2)^4$; otherwise we had an even split, which including rounding has $F_4^{\text{even-split}} < 1$. Thus

$$F_4^{\text{query}} = p_{\text{collect}} F_4^{\text{collect}} + (1 - p_{\text{collect}}) F_4^{\text{even-split}} = m^3/2^4 - O(m^2).$$

We know the combined distribution is 3-independent, so

$$\begin{aligned} P_{(4,0)}^{\text{query}} &= F_4^{\text{query}}/m^{\bar{4}} + C_3^{\text{random}} \\ &= (m^3/2^4 - O(m^2))/m^{\bar{4}} + 1/2^4(1 - (6m^2 - 5m)/m^{\bar{4}}) \\ &= 1/2^4(1 + (m^3 - O(m^2))/m^{\bar{4}})) \\ &= 1/2^4(1 + 1/m - O(1/m^2)). \end{aligned}$$

Next we design a 3-independent scheme with a small skew putting $m/2 + \delta$ on one side and $m/2 - \delta$ on the other side. Here $\delta \approx \sqrt{m/4}$ is rounded up to make $m/2+\delta$ integer. Now $F_2^{\text{small-skew}} = \delta^2 = m/4+O(\sqrt{m})$. We want a second moment of exactly $m/4$. To get rid of the $O(\sqrt{m})$ term, we do an even split with probability $O(1/\sqrt{m})$. From Lemma 1 it follows that combined distribution is 3-independent. Including the small probability of an even split, we get a 4^{th} moment of

$$F_4^{\text{small-skew}} = \delta^4 = m^2/16 \pm O(m^{3/2}).$$

We know the combined distribution is 3-independent, so

$$\begin{aligned} P_{(4,0)}^{\text{small-skew}} &= F_4^{\text{small-skew}}/m^{\bar{4}} + C_3^{\text{random}} \\ &= (m^2/2^4 \pm O(m^{3/2}))/m^{\bar{4}}) + 1/2^4(1 - (6m^2 - 5m)/m^{\bar{4}}) \\ &= 1/2^4(1 - (5m^2 \pm O(m^{3/2}))/m^{\bar{4}})) \\ &= 1/2^4(1 - 5/m^2 \pm O(1/m^{5/2})). \end{aligned}$$

We would get a 4-independent combination of our small skew scheme and our previous query interval scheme if we pick the query scheme with a the probability p satisfying

$$(1-p)P_{(4,0)}^{\text{small-skew}} + pP^{\text{query}} = 1/2^4$$

implying that $p = 5/m \pm O(1/m^2)$. However, this balancing would ignore the special role of the query interval on which we always apply the query scheme.

We now return to the full combination including the special query interval. The distributions happen 4-independently level by level. If at some level we see we have an unlikely "bad" distribution, we can finish the remaining levels with purely random distributions. As long as the overall probability of a bad event is less than $1/2$, this will not affect our expected $\Theta(\lg n)$ query time.

On any level i, the expected number of keys is $m_i = n/2^i$. An interval is fair if it has $m_i \pm 10\sqrt{m_i}$ keys. Recall that for the first $i_0 = \lceil 2/3 \log_2 n \rceil$ levels, we just did perfectly random recursive distributions, which in particular are 4-independent. With high probability, all intervals are fair with $\Theta(n^{1/3})$; otherwise we are in a rare bad event.

On each remaining levels $i \leq 0.9 \log_2 n$, on any fair interval, we will either use the query scheme or the small skew scheme. However, if the query interval is fair, we always use the query scheme. We note that if an interval is fair, then the subintervals will be good except if we collect all keys on one side with the query scheme. In particular, our small skew cannot destroy fairness. Also, recall that if ever the query interval got collected on one side, we continued with fully-random distributions on the remaining levels. Also, recall that even if all ancestor of an interval used the query scheme, then the probability of a one-sided collection is $0(1/m_i)$. We can therefore dismiss it as a bad event if more than a fraction $O(1/\sqrt{m_i})$ of the intervals are not fair. With all this nice structure, we can bound the probability p_{query} that a given quadruple of stored keys is in the query interval. We have roughly the same number of keys in each fair interval, hence roughly the same number of quadruples (within a factor $1 \pm O(\sqrt{1/m_i})$ where $m_i = \Omega(n^{1/10})$). We know the query interval is fair, and we know we have $(1-o(1)n/m_i)$ other intervals, so $p_{\text{query}} = O(m_i/n) = O(1/m_i^2)$. When the query scheme is applied to the query interval, it has a given probability $P_4^{\text{query}}[\text{query}]$ of collecting the quadruple. Hence, for a non-query intervals with $m \approx m_i$ keys, we need the collection probability to be

$$P_4' = (P_4^{\text{random}} - p_{\text{query}} P_4^{\text{query}}[\text{query}])/(1 - p_{\text{query}}) = (1/2^4 - O(1/m_i^2)).$$

Thus for the fair non-query intervals, we should use the query scheme with a probability p' satisfying

$$(1-p')P_{(4,0)}^{\text{small-skew}} + p'P^{\text{query}} = P_4'$$

which is easily possible because $P_{(4,0)}^{\text{small-skew}} < P_4'$.

B Multiplication-Shift Schemes

We show that the simplest and fastest known universal hashing schemes have bad expected performance when used for linear probing on some of the most realistic structured data. This result is inspired by negative experimental findings from [19]. The essential form of the schemes considered have the following basic form: we want to hash ℓ_{in}-bit keys into ℓ_{out}-bit indices. Here $\ell_{in} \geq \ell_{out}$, and the indices are used for the linear probing array. For the typical case of a half full table, we have $2^{\ell_{out}} = m \approx 2\log_2 n$.

Depending on details of the scheme, for some $\ell \geq \ell_{in}, \ell_{out}$, we pick a random multiplier $a \in [2^\ell]$, and compute

$$h_a(x) = (ax \bmod 2^\ell) \div 2^{\ell - \ell_{out}}$$

If $\ell \in \{8, 16, 32, 64\}$, the mod-operation is performed automatically as discarded overflow. The $\div$ operation is just a right shift by $s = \ell - \ell_{out}$, so in C we get the simple code `(a*x)>>s` and the cost is dominated by a single multiplication. For the plain universal hashing in [7], it suffices that $\ell \geq \ell_{in}$ but then a should be odd. For 2-independent hashing as in [6], we need $\ell \geq \ell_{in} + \ell_{out} - 1$. Also we need to add a random number b, but as we shall discuss in the end, these details have no essential impact on the derivation below.

Our basic bad example will be where the keys form the dense interval $[n] = \{0, ..., n-1\}$. However, the problem will not go away this interval is shifted or not totally full, or replaced by an arithmetic progression.

When analyzing the scheme, it is convenient to first consider it as first a mapping into the unit interval $[0, 1)$ via

$$h^0_a(x) = (ax \bmod 2^\ell)/2^\ell.$$

Then $h_a(x) = \lfloor h^0_a(x) 2^{\ell_{out}} \rfloor$. We think of the unit interval as circular, and for any $x \in [0, 1)$, we define

$$\|x\| = \min\{x \bmod 1, -x \bmod 1\}.$$

This is really the absolute value of x considering numbers above $1/2$ negative.

The basic negative example is in itself very simple. We consider the hashing of $[n]$. Suppose for some $x \leq n$ that $\|h^0_a(x)\| \leq 1/(2m)$. The case is illustrated in Figure 1. Then for each $k \in [x]$, the $q = \lfloor n/x \rfloor$ keys y from $[n]$ with $y \bmod x = k$ map to an interval of length $(q-1)/(2m)$, which means that the h_a spreads them on at most $\lceil q/2 \rceil + 1$ consecutive array locations. This interval of locations is roughly double full so the keys in them have an average insert cost of $\Omega(q) = \Omega(n/x)$. However, the same clustering holds for every other $z \in [x]$, that is, the roughly n/x keys y from $[n]$ with $y \bmod x = z$ map to an interval of $\Omega(n/x)$ double full buckets, leading to an average insert cost of $\Omega(n/x)$ per key. The above average costs only measures the interaction between keys from the same equivalence class modulo x. If some of these classes overlap, the average cost will only grow. Since every key is in an equivalence class with average cost $\Omega(n/x)$, hence that this lower bound holds on the average for all keys.

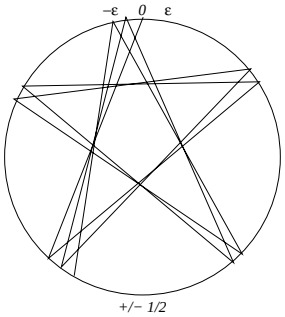

Fig. 1. Case where $h_a^0(5) \leq \varepsilon$.

Note that $\|h_a^0(x)\| \leq 1/(2m)$ implies that $h_a^0(x)$ is contained in an interval of size $1/m$. From the universality arguments of [7, 6] we know that the probability of this event is roughly $1/m$ (we shall return with an exact statement later). We would like to conclude that the expected average cost is $\sum_{x=1}^{n} \Omega(n/x)/m = \Omega(\lg n)$. The answer is correct, but the calculation cheats in the sense that we may have many different x such that $\|h_a^0(x)\| \leq 1/(2m)$, and they should not all be added up.

For any a, let μ_a denote the minimal value such $\|h_a^0(\mu_a)\| \leq 1/(2m)$. For a correct accounting, we only pay the average cost of at $\Omega(n/x)$ if $x = \mu_a$. We will be dealing with negative and positive contributions of similar sizes, so we will have to be careful in the use O-notation. We define $C(x) = n/x$. Then our expected average cost is lower bounded by

$$\Omega\left(\sum_{x=1}^{n} C(x) \Pr_a[x = \mu_a]\right). \tag{1}$$

We are going to lower bound $\sum_{x=1}^{n} C(x) \Pr_a[x = \mu_a]$ without use of O/Ω-notation.

Lemma 3. *Consider any $x \leq n$ such that $\|h_a^0(x)\| \leq 1/(2m)$. Then $x \neq \mu_a$ if and only if for some prime factor p of x, $\|h_a^0(x/p)\| \leq 1/(2pm)$.*

Proof. The "if" part is trivial. For any integer $i \leq m$, we have $\|h_a^0(i\mu_a)\| = i\|h_a^0(\mu_a)\|$, so $\|h_a^0(i\mu_a)\| \leq 1/(2m) \iff \|h_a^0(\mu_a)\| \leq 1/(2im)$. Moreover, if y is not a multiple of μ_a, then $\|h_a^0(y)\| \leq 1/(2m)$ requires that $y \geq 2m$. This is because we will not get $\|h_a^0(y)\| \leq 1/m$ before we have filled the cyclic unit interval with values that are at most $1/(2m)$ apart (c.f. Figure 1). However, $m \geq n$, and we only consider keys $x \leq n$. Thus it suffices to consider the above case where $x = i\mu_a$ and $\|h_a^0(\mu_a)\| \leq 1/(2im)$. Let p be any prime factor of i. Then $\|h_a^0(i/p\,\mu_a)\| = i/p\,\|h_a^0(\mu_a)\| \leq 1/(2pm)$.

Note that $\|h_a^0(x)\| \leq \varepsilon$ implies that $h_a^0(x)$ is contained in an interval of size 2ε. To illustrate the basic argument, we first assume a false but simplifying perfect

uniformity on $h_a^0(x)$ for random a:

$$\Pr_a[\|h_a^0(x)\| \leq \varepsilon/2] = \varepsilon. \tag{2}$$

Now

$$\begin{aligned}\Pr_a[x = \mu_a] &\geq \Pr_a[\|h_a^0(x)\| \leq 1/(2m)] - \sum_{p \text{ prime factor of } i} \Pr_a[\|h_a^0(x/p)\| \leq 1/(2pm)] \\ &> (1 - \sum_{p \text{ prime factor of } x} 1/p)/m \end{aligned} \tag{3}$$

We note that the lower-bound (3) may be very negative since there are values of x for which $\sum_{p \text{ prime factor of } x} 1/p = \Theta(\lg \lg x)$. Nevertheless (3) suffices with an appropriate reordering of terms, as described below:

$$\begin{aligned}\sum_{x=1}^{n} C(x) \Pr_a[x = \mu_a] &= \sum_{x=1}^{n} \left(n/(xm) \left(1 - \sum_{p \text{ prime factor of } x} 1/p \right) \right) \\ &\geq \sum_{x=1}^{n} \left(n/(xm) \left(1 - \sum_{\text{prime } p=2,3,5,..} 1/p^2 \right) \right)\end{aligned}$$

Above we simply moved terms of the form $-n/(xmp)$ where p is a prime factor of x to $x' = x/p$ in the form $-n/(x'mp^2)$. Conservatively, we include $-n/(x'mp^2)$ for all primes p even if $px' > n$. Since $\sum_{\text{prime } p=2,3,5,..} 1/p^2 < 0.453$, we get

$$\begin{aligned}\sum_{x=1}^{n} C(x) \Pr_a[x = \mu_a] &> \sum_{x=1}^{n} 0.547n/(xm) \\ &= \Omega(n/m \lg n).\end{aligned}$$

We would now be done if $h_a(x)$ was uniform and satisfied the equality (2), which is false. For every $\varepsilon < 1/2$, we will establish

$$\Pr[\|h_a^0(x)\| \leq \varepsilon/2] \leq 2\varepsilon \tag{4}$$

Moreover, we will have:

$$\Pr[\|h_a^0(x)\| \leq 1/(2m)] \geq 1/m. \tag{5}$$

To satisfy (4)-(5), we will only consider cases where both x and a are odd. Assuming that the random number a is odd can only double the expected cost, and in fact, this is already part of the plain universal hashing from [7]. When x is odd and a is a uniformly distributed odd ℓ-bit number, then $a * x \bmod 2^\ell$ is uniformly distributed odd random number. To get $h_a^0(x)$, we divide by 2^ℓ, turning the number into a fraction, and now we have a uniform distribution on the odd multiples of $1/2^\ell$. As to extreme cases, we note that $\Pr[\|h_a^0(x)\| < 1/2^\ell] = 0$ while $\Pr[\|h_a^0(x)\| \leq 1/2^\ell] = 4/2^\ell$. Generally, for any $\varepsilon < 1/2$, we have $\Pr[\|h_a^0(x)\| \leq$

$\varepsilon/2] \le 2\varepsilon$, as stated in (4). Our distribution is uniformly distributed on the centers of the intervals $[i/2^{\ell-1}, (i+1)/2^{\ell-1}]$. Therefore $\Pr[\|h_a^0(x)\| \le \varepsilon/2] = \varepsilon$ whenever $\varepsilon = 2^{j-\ell}$ for some integer $j \ge 2$. We also had $\Pr[\|h_a^0(x)\| \le \varepsilon/2] = 2\varepsilon$ for $\varepsilon = 2/2^\ell$. Recall that $m = 2^{\ell_{out}}$. Hence (5) follows when

$$\ell_{out} < \ell. \tag{6}$$

Using the true statements (4) and (5), we recompute the expected cost. Note that even though we only consider odd $x = \mu_a$, we still hash all the odd and even keys in $[0, n)$, so our cost $C(x) = n/x$ is unchanged. Using that the prime 2 cannot divide an odd x, we get

$$\begin{aligned}
\sum_{\text{odd } x=1}^{n} C(x) \Pr_a[x = \mu_a] &= \sum_{\text{odd } x=1}^{n} \left(n/(xm) \left(1 - 2 \sum_{p \text{ prime factor of } x} 1/p \right) \right) \\
&\ge \sum_{\text{odd } x=1}^{n} \left(n/(xm) \left(1 - 2 \sum_{\text{prime } p>2} 1/p^2 \right) \right) \\
&\ge \sum_{\text{odd } x=1}^{n} 0.595 n/(xm) \\
&= \Omega(n/m \lg n).
\end{aligned}$$

This completes our lower-bound proof for the plain universal hashing from [7] when $\ell_{out} < \ell$. Note that if $\ell_{out} = \ell$ then the plain universal hashing scheme with an odd multiplier is just a permutation, and then linear probing will work perfectly.

For the 2-universal hashing [6] there are two differences. One is that the multiplier may also be even, but restricting it to be odd can only double the cost. The other difference is that we add an additional ℓ-bit parameter b, yielding a scheme of the form:

$$h_{a,b}(x) = \lfloor ((ax + b) \bmod 2^\ell)/2^{\ell - \ell_{out}} \rfloor.$$

The only effect of b is a cyclic shift of the double full buckets, and this has no effect on the linear probing cost. For the 2-independent hashing, we have $\ell \ge \ell_{in} + \ell_{out} - 1$, so (6) is trivially satisfied. Hence again we have an expected average linear probing cost of $\Omega(n/m \lg n)$.

Finally, we sketch some variations of our bad input. Currently, we just considered the set $[n]$ of input keys, but it makes no essential difference if instead for some integer constants α and β, we consider the input set $X = \alpha[n] + \beta = \{i\alpha + \beta \mid x \in [n]\}$ for . The β is just adds a cyclic shift like the b in 2-independent hashing. The α is essentially absorbed in the random multiplier a. What we get now is that if for some $x \in [n]$, we have $\|h_a^0(\alpha x)\| \le 1/(2m)$, then again we get an average cost $\Omega(n/x)$. A consequence is that no odd multiplier a is universally safe because there always exists an inverse α (with $a\alpha \bmod 2^\ell = 1$) leading to

a linear cost. It is not hard to also construct bad examples for even multipliers since they essentially just drop some of the least significant bits.

Another more practical concern is if the input set X is an ε-fraction of $[n]$. Our worry now is if for some $x \leq n$ that $\|h_a^0(x)\| \leq \varepsilon/(2m)$. In that case, for each $k \in [x]$, the $q = \lfloor n/x \rfloor$ potential keys y from $[n]$ with $y \bmod x = k$ map would map to an interval of length $\varepsilon(q-1)/(2m)$. This means that h_a spreads these potential keys on at most $\lceil \varepsilon q/2 \rceil + 1$ consecutive array locations. A ε-fraction of these keys are real, so on the average, these intervals become double full, leading to an average cost of $\Omega(\varepsilon n/x)$. Strengthening (6) to assume $\varepsilon \geq 2^{\ell_{out}-\ell}$, we essentially get that all probabilities are reduced by ε. Thus we end with a cost of $\Omega(\varepsilon^2 n/m \lg n) = \Omega(\varepsilon|X|/m \lg n)$.

C Minwise Independence

We now consider the lack of minwise independence with a hashing scheme

$$h_{a,b}(x) = ((ax+b) \bmod 2^\ell).$$

Shifting out less significant bits does not make much sense since we are not hashing to entries in an array. The analysis will be very similar to the one done for linear probing in Section B, and we will only sketch it. Now the added b is necessary to get anything meaningful, for without it, zero would always get the minimal hash value $h_{a,0}(0) = 0$. The effect of adding b is to randomly spin the wheel from Figure 1. As in Section, it is convenient to divide by 2^ℓ to get fractions in the cyclic unit interval. We thus define $h_{a,b}^0(x) = h_{a,b}(x)/2^\ell$.

The bad case will be the interval $[n]$ versus a random query key q. We assume that n is a power of two. To see the parallels to Section B, think of $m = n$. For any a, we define $\mu_a > 0$ to be the smallest value such that $h_{a,0}^0(\mu_a) \leq 1/(2n)$. Then $h_{a,0}^0([n])$ falls in μ_a equidistant intervals, each of length at most $1/(2\mu_a)$. This leaves us with μ_a equidistant empty intervals, each of length at least $\leq 1/(2\mu_a)$. Together these empty intervals cover half the cyclic unit interval. When we add b it has the effect of placing 0 randomly on the cycle. Having chosen a and b, a random q is also hashed to random place on the cycle. The probability that q and 0 end in the same empty interval and with q after 0 in the interval is $1/(2^3\mu_a)$. In this case, q has the minimal hash value, but that should only have happened with probability $1/n$. Thus, relatively speaking, the probability is to high by a factor $\Omega(n/\mu_a)$, matching the linear probing cost from Section B. As a result we will also end up concluding that the expected min-probability is to high by a factor $\Omega(\lg n)$.

As in Section B, there are some details to consider. It is convenient to restrict ourselves to odd values of a, b, $x = \mu_a$ and q. As a result, all our hash values are odd, and at odd multiples of $1/2^\ell$ in the cyclic unit interval. The analysis goes through as long as $\ell > \lceil \log_2 n \rceil$, and in fact, we can shit our all but the $\lceil \log_2 n \rceil$ most significant bits and yet have the same lower bound that the min-probability is too high by a factor $\Omega(\lg n)$.