LINEAR PROBING WITH 5-WISE INDEPENDENCE*

ANNA PAGH†, RASMUS PAGH†, AND MILAN RUŽIĆ†

Abstract. Hashing with linear probing dates back to the 1950s, and is among the most studied algorithms for storing (key,value) pairs. In recent years it has become one of the most important hash table organizations since it uses the cache of modern computers very well. Unfortunately, previous analyses rely either on complicated and space consuming hash functions, or on the unrealistic assumption of free access to a hash function with random and independent function values. Carter and Wegman, in their seminal paper on universal hashing, raised the question of extending their analysis to linear probing. However, we show in this paper that linear probing using a 2-wise independent hash function may have expected *logarithmic* cost per operation. Recently, Pătraşcu and Thorup have shown that also 3- and 4-wise independent hash functions may give rise to logarithmic expected query time.

On the positive side, we show that 5-wise independence is enough to ensure constant expected time per operation. This resolves the question of finding a space and time efficient hash function that provably ensures good performance for hashing with linear probing.

Key words. Hashing, Linear Probing

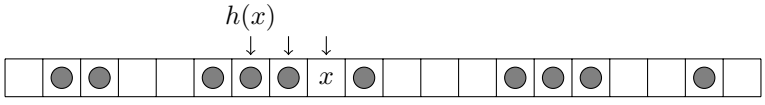
AMS subject classifications. 68P10, 68W40, 68W20

1. Introduction.

One if the most common subtasks in computer programs is to identify the location in memory of a particular piece of information, identified by a unique key. As an example, consider a program that counts the number of occurrences of words in a document. For each word, the counter associated with that word either needs to be created (if the word is seen for the first time), or located in memory. Naïve solutions to this problem would give solutions that scale poorly to settings with many keys (documents with many distinct words, in the example). Many kinds of data intensive software such as database systems, data stream analysis algorithms, and network routers, rely on key lookup as a crucial component.

Fortunately, there exist methods that allow insertion of new keys as well as retrieval of existing keys to be done in time *independent* of the size of the key set. The common feature of these methods is that they use one or more randomly chosen functions (so-called *hash functions*) in conjunctions with an array (or vector), referred to as the *hash table*, that is able to hold keys and associated data. In many cases, a programmer using a hash table, rather than a naïve solution, can reduce the running time of an algorithm by a factor proportional to the size of the data set handled.

Hashing with linear probing is perhaps the simplest method for organizing a hash table. Given a hash function h, a key x is inserted in the hash table by searching for the first vacant position in the sequence $h(x), h(x)+1, h(x)+2, \ldots$ (Here, addition is modulo r, the size of the hash table.) Retrieval of a key proceeds similarly, until either the key is found, or a vacant position is encountered, in which case the key is not present in the hash table. Below we have illustrated insertion (and retrieval) of a key x in a hash table, where other keys are shown as grey balls.



*This paper is based on [9] that previously appeared in SIAM Journal on Computing.

†IT University of Copenhagen, Denmark.

An implicit assumption made here is that a key fits in a single hash table location — for generalization to variable length keys see [17]. Deletion of keys can be performed by moving keys back in the probe sequence in a greedy fashion (ensuring that no key x is moved to before $h(x)$), until no such move is possible (when a vacant array position is encountered).

Linear probing is attractive compared to other hashing methods because it accesses memory in a sequential fashion. The memory systems of modern computers are optimized for this type of access pattern, which means that inspecting several consecutive memory locations does not take significantly more time than inspecting a single, randomly chosen memory location. In fact, experimental studies [1, 5, 10] have found linear probing to be the fastest hash table organization for hash tables that are moderately filled (30-70%). While linear probing operations are known to require more instructions than those of other hashing methods, the more efficient memory access pattern makes it very fast in practice.

1.1. Early analysis and heuristic implementations. Linear probing dates back to 1954, but was first analyzed by Knuth in a 1963 memorandum [6] now considered to be the birth of the area of analysis of algorithms [11]. Even if, say, half of the hash table is empty, it is not clear a priori that it will be fast to find a vacant location for a new key x. The reason is that $h(x)$ may lie in a long interval of occupied positions. In fact, if such an interval starts to form there will be a "pileup" phenomenon, since the expected number of new keys added to the interval is proportional to its length.

Knuth's analysis, showing that the expected time per operation is independent of the number of keys, is based on the assumption that h has uniformly distributed and independent function values. But actually representing such a function is not a viable option: Retrieving the random value $h(x)$ associated with x seems as hard as the problem we wanted to solve in the first place! Therefore, such analyses do not say much about practical, implementable solutions.

The hash functions used to implement linear probing in practice are heuristics, and there is no known theoretical guarantee on their performance. Since linear probing is particularly sensitive to a bad choice of hash function, Heileman and Luo [5] advice *against* linear probing for general-purpose use.

1.2. Analysis using limited randomness. In 1977, Carter and Wegman's notion of universal hashing [3] initiated a new era in the design of hashing algorithms, where explicit and efficient ways of choosing hash functions replaced the assumption of full randomness. The big insight was that in many cases it is sufficient that the hash function is random with respect to small sets of keys. For example, consider the function $h(x) = (ax + b) \bmod p$, where p is a fixed prime number and a, b are chosen independently at random from $\{0, \ldots, p-1\}$. For any two distinct integers $x_1, x_2 \in \{0, \ldots, p-1\}$ the values $h(x_1)$ and $h(x_2)$ are independent and uniformly random (over the choice of a and b). This "2-wise independence" turns out to be sufficient to guarantee many of the properties possessed by fully random hash functions [3, 20]. More generally, researchers have considered *k-wise independence* where any k values of the hash function are independent.

In their seminal paper, Carter and Wegman state it as an open problem to "Extend the analysis to [...] double hashing and open addressing."[1] The first analysis of

[1]Nowadays the term "open addressing" refers to any hashing scheme where the data structure is an array containing only keys and empty locations. However, Knuth used the term to refer to linear

linear probing relying only on limited randomness was given by Siegel and Schmidt in [14, 16]. Specifically, they show that $O(\log n)$-wise independence is sufficient to achieve essentially the same performance as in the fully random case. (We use n to denote the number of keys inserted into the hash table.) Another paper by Siegel [15] shows that evaluation of a hash function from a $O(\log n)$-wise independent family requires time $\Omega(\log n)$ unless the space used to describe the function is $n^{\Omega(1)}$. A family of functions is given that achieves space usage n^ϵ and constant time evaluation of functions, for any $\epsilon > 0$. However, this result is only of theoretical interest since the associated constants are very large. A construction with similar properties but smaller constants has later been given by Dietzfelbinger and Weidling [4].

A significant drawback of both methods, besides rather complex function evaluation, is the use of random accesses to the memory locations holding the hash function description. This means that we lose the advantage of accessing memory in a sequential fashion!

1.3. Our results. We show in this paper that linear probing using a 2-wise independent family may have expected *logarithmic* cost per operation. Specifically, we resolve the open problem of Carter and Wegman by showing that linear probing insertion of n keys in a table of size $2n$ using a function of the form

$$x \mapsto ((ax + b) \bmod p) \bmod 2n,$$

where $p = 4n + 1$ is prime and we randomly choose $a \in [p]\backslash\{0\}$ and $b \in [p]$, requires $\Omega(n \log n)$ insertion steps in expectation for a worst case insertion sequence (chosen independently of a and b). Since the total insertion cost equals the total cost of looking up all keys, the expected average time to look up a key in the resulting hash table is $\Omega(\log n)$. The main observation behind the proof is that if a is the multiplicative inverse (modulo p) of a small integer m, then inserting a certain set that consists of two intervals has expected cost $\Omega(n^2/m)$.

On the positive side, we show that *5-wise independence* is enough to ensure constant expected time per operation, when the *load factor* $\alpha \stackrel{\text{def}}{=} n/r$ is bounded away from 1. Our proof is based on a new way of bounding the cost of linear probing operations, by counting intervals in which "many" probe sequences start. When beginning this work, our first observation was that a key x can be placed in location $h(x) + l \bmod r$ only if there is an interval $I \ni h(x)$ where $|I| \geq l$ and there are $|I|$ keys from S with hash value in I. A slightly stronger fact is shown in Lemma 4.1. Since the expected number of hash values in an interval I is $\alpha|I|$, long such intervals are "rare" if the hash function exhibits sufficiently high independence.

The analysis in this version of the paper shows a bound of $O(\frac{1}{(1-\alpha)^{5/2}})$ expected time per operation at load factor α. Using similar, but more involved ideas it is possible to replace the exponent 5/2 by 13/6 [9], and even 2 [13]. The latter exponent is identical to that obtained by a fully random hash function [6].

1.4. Practical implications. Our results imply that simple and efficient hash functions, whose description can be stored in CPU registers, can be used to give provably good expected performance. For completeness we briefly describe concrete ways of chosing hash functions.

---

probing in [6], and since it is mentioned here together with the double hashing probe sequence, we believe that it refers to linear probing.

Polynomial hash functions. Carter and Wegman [19] observed that the family of degree $k-1$ polynomials in any finite field is k-wise independent. Specifically, for any prime p we may use the field defined by arithmetic modulo p to get a family of functions from $[p]$ to $[p]$ where a function can be evaluated in time $O(k)$ on a RAM, assuming that addition and multiplication modulo p can be performed in constant time. To obtain a smaller range $R = [r]$ we may map integers in $[p]$ down to R by a modulo r operation. This of course preserves independence, but the family is now only close to uniform. Specifically, the maximum load $\bar{\alpha}$ for this family is in the range $[\alpha,\ (1 + r/p)\alpha]$. By choosing p much larger than r we can make $\bar{\alpha}$ arbitrarily close to α.

Tabulation-based hash functions. A k-wise independent family proposed by Thorup and Zhang [18] has uniformly distributed function values in $[r]$, and thus $\bar{\alpha} = \alpha$. The construction for 5-wise independence is particularly appealing when keys are short (e.g., 32 bits). If we interpret a key as a tuple of two numbers (x_1, x_2) the hash function $h(x_1, x_2) = (h_1(x_1) + h_2(x_2) + h_3(x_1 + x_2)) \bmod r$ is 5-wise independent assuming h_1, h_2, h_3 are. If x_1, x_2, and $x_1 + x_2$ are small numbers (e.g., 16 bits) we can use a lookup table to store all hash values. This means that $h(x_1, x_2)$ can be evaluated using three table lookups and some additions. If the tables are small enough to fit within the CPU cache, evaluation is particularly efficient. In fact, this construction makes k-wise independence truly competitive with popular heuristics, for small $k > 3$, in terms of evaluation time [18].

1.5. Subsequent work. The work of the present paper has been built upon in designing hash tables with additional considerations. Blelloch and Golovin [2] described a linear probing hash table implementation that is *strongly history independent*. Thorup [17] studied how to get efficient compositions of hash functions for linear probing when the domain of keys is complex, like the set of variable-length strings. Most recently, Pătraşcu and Thorup showed that linear probing works even with the simplest possible type of tabulation-based hashing [13].

On the lower bound side, Pătraşcu and Thorup showed that there exist 3- and 4-wise independent hash function constructions that result in logarithmic time per operation [12]. This means that there is no hope to improve our results to require lower independence. They also show even worse performance for single operations under 2-wise independence than exhibited in this paper.

2. Preliminaries. Let $[x] \stackrel{\text{def}}{=} \{0, 1, \ldots, x-1\}$. Throughout this paper S denotes a subset of some universe U, and h will denote a function from U to $R \stackrel{\text{def}}{=} [r]$. We denote the elements of S by $\{x_1, x_2, \ldots, x_n\}$, and refer to the elements of S as *keys*. We let $n \stackrel{\text{def}}{=} |S|$, and $\alpha \stackrel{\text{def}}{=} n/r$.

A family $\mathcal{H}$ of functions from U to R is k-wise independent if for any k distinct elements $x_1, \ldots, x_k \in U$ and h chosen uniformly at random from $\mathcal{H}$, the random variables $h(x_1), \ldots, h(x_k)$ are independent[2]. We refer to the variable

$$\bar{\alpha}_{\mathcal{H}} \stackrel{\text{def}}{=} n \max_{x \in U,\, \rho \in R} \Pr_{h \in \mathcal{H}}\{h(x) = \rho\}$$

as the *maximum load* of $\mathcal{H}$. When the hash function family in question is understood

[2]We note that in some papers, the notion of k-wise independence is stronger in that it is required that function values are uniformly distributed in R. However, some interesting k-wise independent families have a slightly nonuniform distribution, and we will provide analysis for such families as well.

from the context, we omit the subscript of $\bar{\alpha}$. If $\mathcal{H}$ distributes hash function values of all elements of U uniformly on R, we will have $\bar{\alpha} = \alpha$, and in general $\bar{\alpha} \geq \alpha$.

For $Q \subseteq R$ we introduce notation for the "translated set"

$$a + Q \stackrel{\text{def}}{=} \{(a + y) \bmod r \mid y \in Q\} \ .$$

An *interval* (modulo r) is a set of the form $a + [b]$, for integers a and b. When we write $[a - b, a)$ this interval represents the set $a - 1 - [b]$. We will later use sets of the form $h(x) + Q$, for a fixed x and with Q being an interval.

2.1. A probabilistic lemma. Here we state a lemma that is essential for our upper bound results, described in Section 4. It gives an upper bound on the probability that an interval around a particular hash function value contains the hash function values of "many" keys. The proof is similar to the proof of [8, Lemma 4.19].

LEMMA 2.1. *Let $S \subseteq U$ be a set of size n, and $\mathcal{H}$ a 5-wise independent family of functions from U to R with maximum load at most $\bar{\alpha} < 1$. If h is chosen uniformly at random from $\mathcal{H}$, then for any $Q \subset R$ of size q, and any fixed $x \in U \setminus S$,*

$$\Pr\left\{\left|\{y \in S \ : \ h(y) \in (h(x) + Q)\}\right| \geq \bar{\alpha}q + d\right\} < \frac{4\bar{\alpha}q^2}{d^4} \ .$$

Proof. Denote by A the event that $\left|\{y \in S \ : \ h(y) \in (h(x) + Q)\}\right| \geq \bar{\alpha}q + d$. We will show a stronger statement, namely that the same upper bound holds for the conditional probability $\Pr\{A \mid h(x) = \rho\}$, for any $\rho \in R$. Notice that the subfamily $\{h \in \mathcal{H} \mid h(x) = \rho\}$ is 4-wise independent on $U \setminus \{x\}$, and that the distribution of function values is identical to the distribution when h is chosen from $\mathcal{H}$. The statement of the lemma will then follow from

$$\Pr(A) = \sum_{\rho \in R} \Pr\{h(x) = \rho\}\Pr\{A \mid h(x) = \rho\} < r \cdot \frac{1}{r}\frac{4\bar{\alpha}q^2}{d^4} \ .$$

Let $p_i \stackrel{\text{def}}{=} \Pr\{h(x_i) \in (h(x) + Q)\}$, and consider the random variables

$$X_i \stackrel{\text{def}}{=} \begin{cases} 1 - p_i, & \text{if } h(x_i) \in h(x) + Q \\ -p_i, & \text{otherwise} \end{cases} \ .$$

Let $X \stackrel{\text{def}}{=} \sum_i X_i$ and observe that

$$\left|\{y \in S \ : \ h(y) \in (h(x) + Q)\}\right| = X + \sum_i p_i \leq X + \bar{\alpha}q \ .$$

The last inequality above is by the definition of maximum load. So to prove the lemma it suffices to bound $\Pr\{X \geq d\}$. We will use the 4th moment inequality

$$\Pr\{X \geq d\} \leq \mathrm{E}(X^4)/d^4 \ .$$

Clearly, $\mathrm{E}(X_i) = 0$ for any i, and the variables $X_1, \ldots, X_n$ are 4-wise independent. Therefore we have $\mathrm{E}(X_{i_1}X_{i_2}X_{i_3}X_{i_4}) = 0$ unless $i_1 = i_2 = i_3 = i_4$ or (i_1, i_2, i_3, i_4) contains 2 numbers, both of them exactly twice. This means that

$$\begin{aligned} \mathrm{E}(X^4) &= \sum_{1 \leq i_1, i_2, i_3, i_4 \leq n} \mathrm{E}(X_{i1}X_{i2}X_{i3}X_{i4}) \\ &= \sum_{1 \leq i \leq n} \mathrm{E}(X_i^4) + \sum_{1 \leq i < j \leq n} \tbinom{4}{2}\mathrm{E}(X_i^2)\mathrm{E}(X_j^2). \end{aligned}$$

The first sum can be bounded as follows:

$$\begin{aligned}\sum_i \mathrm{E}(X_i^4) &= \sum_i (p_i(1-p_i)^4 + (1-p_i)p_i^4) \\ &= \sum_i p_i(1-p_i)((1-p_i)^3 + p_i^3) \\ &< \sum_i p_i \leq \bar{\alpha}q \ .\end{aligned}$$

The second sum is:

$$\begin{aligned}\sum_{1\leq i<j\leq n} 6(p_i(1-p_i))(p_j(1-p_j)) &< 3 \sum_{1\leq i,j\leq n} p_i p_j \\ &= 3(\sum_i p_i)^2 \leq 3(\bar{\alpha}q)^2 \ .\end{aligned}$$

In conclusion we have

$$\Pr\{X \geq d\} \leq \mathrm{E}(X^4)/d^4 < \frac{3(\bar{\alpha}q)^2 + \bar{\alpha}q}{d^4} < \frac{4\bar{\alpha}q^2}{d^4} \ ,$$

finishing the proof. □

3. 2-wise independence. In this section we show that 2-wise independence is not sufficient to ensure good performance for linear probing: Logarithmic time per operation is needed for a worst-case set. This complements our upper bounds for 5-wise (and higher) independence. We will consider two 2-wise independent families: The first one is a very commonly used hash function family. The latter family is similar to the first, except that we have ensured function values to be uniformly distributed in R. To lower bound the cost of linear probing we use the following lemma.

LEMMA 3.1. *Suppose a set S of n keys is inserted in a linear probing hash table of size $r > n$. Let $\{S_j\}_{j=1}^{\ell}$ be any partition of S such that for every set S_j the set $I_j \stackrel{def}{=} h(S_j)$ is an interval (modulo r), and $|I_j| \leq r/2$. Then the total number of steps to perform the insertions is at least*

$$\sum_{1\leq j_1<j_2\leq \ell} |I_{j_1} \cap I_{j_2}|^2/2 \ .$$

Proof. We proceed by induction on ℓ. Since the number of insertion steps is independent of the order of insertions [7, p. 538], we may assume that the insertions corresponding to S_ℓ occur last and in left-to-right order of hash values. By the induction hypothesis, the total number of steps to do all preceding insertions is at least

$$\sum_{1\leq j_1<j_2\leq \ell-1} |I_{j_1} \cap I_{j_2}|^2/2 \ .$$

For $1 \leq j_1, j_2 \leq \ell$ let $S_{j_1 j_2}$ denote the set of keys from S_{j_1} that have probe sequences starting in I_{j_2}, i.e $S_{j_1 j_2} = \{x \in S_{j_1} \mid h(x) \in I_{j_2}\}$. For any $x \in S_{\ell j}$ the insertion of x will pass all the elements of $S_{j\ell}$ "after h(x)", i.e., whose hash value is in $h(x) + [r/2]$. This means that at least $|I_j \cap I_\ell|^2/2$ steps are used during the insertions of the keys from S_ℓ to pass locations occupied by keys of S_j. Summing over all $j < \ell$ and adding to the bound from the induction hypothesis yields the desired result. □

3.1. Linear congruential hash functions. We first consider the following family of functions, introduced by Carter and Wegman [3] as a first example of a universal family of hash functions:

$$\mathcal{H}(p,r) \stackrel{\text{def}}{=} \{x \mapsto ((ax+b) \bmod p) \bmod r \mid 0 < a < p,\, 0 \le b < p\}$$

where p is any prime number and $r \le p$ is any integer. Functions in $\mathcal{H}(p,r)$ map integers of $[p]$ to $[r]$.

THEOREM 3.2. *For $r = \lceil p/2 \rceil$ there exists a set $S \subseteq [p]$, $|S| \le r/2$, such that the expected cost of inserting the keys of S in a linear probing hash table of size r using a hash function chosen uniformly at random from $\mathcal{H}(p,r)$ is $\Omega(r \log r)$.*

Proof. We give a randomized construction of S, and show that when choosing h at random from $\mathcal{H}(p,r)$ the expected total insertion cost for the keys of S is $\Omega(r \log r)$. This implies the existence of a fixed set S with at least the same expectation for random $h \in \mathcal{H}(p,r)$. Specifically, we partition $[p]$ into 8 intervals $U_1, \ldots, U_8$, such that $\bigcup_i U_i = [p]$ and $r/4 \ge |U_i| \ge r/4 - 1$ for $i = 1, \ldots, 8$, and let S be the union of two of the sets $U_1, \ldots, U_8$ chosen at random (without replacement). Note that $|S| \le r/2$, as required.

Consider a particular function $h \in \mathcal{H}(p,r)$ and the associated values of a and b. Let $\hat{h}(x) \stackrel{\text{def}}{=} (ax+b) \bmod p$, and let m denote the unique integer in $[p]$ such that $am \bmod p = 1$ (i.e., $m = a^{-1}$ in GF(p)). Since $\hat{h}$ is a permutation on $[p]$, the sets $\hat{h}(U_i)$, $i = 1, \ldots, 8$, are disjoint. We note that for any x, $\hat{h}(x+m) = (\hat{h}(x)+1) \bmod p$. Thus, for any k, $\hat{h}(\{x, x+m, x+2m, \ldots, x+km\})$ is an interval (modulo p) of length $k+1$. This implies that for all i there exists a set $\hat{L}_i$ of m disjoint intervals such that $\hat{h}(U_i) = \bigcup_{I \in \hat{L}_i} I$. Similarly, for all i there exists a set L_i of at most $m+1$ intervals (not necessarily disjoint) such that we have the multiset equality $h(U_i) = \bigcup_{I \in L_i} I$. Since all intervals in $\bigcup_i \hat{L}_i$ are disjoint and their sizes differ by at most 1, an interval in $\bigcup_i L_i$ can intersect at most two other intervals in $\bigcup_i L_i$. We now consider two cases:

1. Suppose there is some i such that

$$\sum_{I_1, I_2 \in L_i, I_1 \ne I_2} |I_1 \cap I_2| \ge r/16 \ . \tag{3.1}$$

   With constant probability it holds that $U_i \subseteq S$. We apply Lemma 3.1 on the set U_i and a partition of U_i that corresponds to the interval collection L_i. The lemma gives us a lower bound of

$$\sum_{I_1, I_2 \in L_i, I_1 \ne I_2} |I_1 \cap I_2|^2/2 \tag{3.2}$$

   on the number of probes made during all insertions. This sum is minimized if all nonzero intersections have the same size. Suppose that there are $k = O(m)$ nonzero intersections. According to (3.1) the equal size of intersections would have to be $\Omega(r/k)$. Therefore the sum in (3.2) is $\Omega(r^2/k) = \Omega(r^2/m)$.
2. Now suppose that for all i,

$$\sum_{I_1, I_2 \in L_i, I_1 \ne I_2} |I_1 \cap I_2| < r/16 \ .$$

   Note that any value in $[r-1]$ is contained in exactly two intervals of $\bigcup_i L_i$. By the assumption, the number of values that occur in two intervals from the

same collection L_i, for any i, is less than $8 \cdot r/16 = r/2$. Thus there exist i_1, i_2, $i_1 \neq i_2$, such that $|h(U_{i_1}) \cap h(U_{i_2})| = \Omega(r)$. With constant probability we have that $S = U_{i_1} \cup U_{i_2}$. We now apply Lemma 3.1. Consider just the terms in the sum of the form $|I_1 \cap I_2|^2/2$, where $I_1 \in L_{i_1}$ and $I_2 \in L_{i_2}$. As before, this sum is minimized if all $O(m)$ intersections have the same size, and we derive an $\Omega(r^2/m)$ lower bound on the number of insertion steps.

For a random $h \in \mathcal{H}(p, r)$, m is uniformly distributed in $\{1, \ldots, p\}$ (the mapping $a \mapsto a^{-1}$ is a permutation of $\{1, \ldots, p\}$). This means that the expected total insertion cost is:

$$\Omega\left(\frac{1}{p}\sum_{m=1}^{p} r^2/m\right) = \Omega\left(\frac{r^2}{p}\log p\right) = \Omega(r \log r) \ .$$

□

3.2. Family with uniform distribution. One might wonder if the lower bound shown in the previous section also holds if the hash function values are uniformly distributed in R. We slightly modify $\mathcal{H}(p, r)$ to remain 2-wise independent and also have uniformly distributed function values. Let $\hat{p} \stackrel{\text{def}}{=} \lceil p/r \rceil\, r$, and define:

$$g(y, \hat{y}) \stackrel{\text{def}}{=} \begin{cases} \hat{y} & \text{if } \hat{y} \geq p \\ y & \text{otherwise} \end{cases} \ .$$

For a vector v let v_i denote the $i+1$st component (indexes starting with zero). We define:

$$\mathcal{H}^*(p, r) \stackrel{\text{def}}{=} \{x \mapsto g((ax + b) \bmod p, v_x) \bmod r \mid 0 \leq a < p,\ 0 \leq b < p,\ v \in [\hat{p}]^p\}$$

LEMMA 3.3 (2-wise independence). *For any pair of distinct values $x_1, x_2 \in [p]$, and any $y_1, y_2 \in [r]$, if h is chosen uniformly at random from $\mathcal{H}^*(p, r)$, then*

$$\Pr\{h(x_1) = y_1 \wedge h(x_2) = y_2\} = 1/r^2 \ .$$

Proof. We will show something stronger than claimed, namely that the family

$$\mathcal{H}^{**} = \{x \mapsto g((ax + b) \bmod p, v_x) \mid 0 \leq a < p,\ 0 \leq b < p,\ v \in [\hat{p}]^p\}$$

is 2-wise independent and has function values uniformly distributed in $[\hat{p}]$. Since r divides $\hat{p}$ this will imply the lemma. Pick any pair of distinct values $x_1, x_2 \in [p]$, and consider a random function $h \in \mathcal{H}^{**}$. Clearly, v_{x_1} and v_{x_2} are uniform in $[\hat{p}]$ and independent. We note as in [3] that for any $y_1', y_2' \in [p]$ there is exactly one choice of a and b that makes $(ax_1 + b) \bmod p = y_1'$ and $(ax_2 + b) \bmod p = y_2'$. This is because the matrix $\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix}$ is invertible. As a consequence, $(ax_1 + b) \bmod p$ and $(ax_2 + b) \bmod p$ are uniform in $[p]$ and independent. We can think of the definition of $h(x)$ as follows: The value is v_x unless $v_x \in [p]$, in which case we substitute v_x for another random value in $[p]$, namely $(ax + b) \bmod p$. It follows that hash function values are uniformly distributed, and 2-wise independent. □

COROLLARY 3.4. *Theorem 3.2 holds also if we replace $\mathcal{H}(p, r)$ by $\mathcal{H}^*(p, r)$. In particular, 2-wise independence with uniformly distributed function values is not a sufficient condition for linear probing to have expected constant cost per operation.*

Proof. Consider the parameters a, b, and v of a random function in $\mathcal{H}^*(p,r)$. Since $r = \lceil p/2 \rceil$ we have $\hat{p} = p+1$, and $(p/\hat{p})^p > 1/4$. Therefore, with constant probability it holds that $a \neq 0$ and $v \in [p]^p$. Restricted to functions satisfying this, the family $\mathcal{H}^*(p,r)$ is identical to $\mathcal{H}(p,r)$. Thus, the lower bound carries over (with a smaller constant). By Lemma 3.3, $\mathcal{H}^*$ is 2-wise independent with uniformly distributed function values. □

We remark that the lower bound is tight. A corresponding $O(n \log n)$ upper bound can be shown by applying the framework of section 4, but using Chebychev's inequality rather than Lemma 2.1 as the basic tool for bounding probabilities (see also [12]).

4. 5-wise independence. We want to bound the expected number of probes into the table made during any single operation (insertion, deletion, or lookup of a key x) when the hash table contains the set S of keys. It is well known that for linear probing, the set P of occupied table positions depends only on the set S and the hash function, independent of the sequence of insertions and deletions performed. An operation on key x makes no more than

$$1 + \max\{l \mid h(x) + [l] \subseteq P\}$$

probes into the table, because the iteration stops when the next unoccupied position is found (or sooner in case of a successful search). We first show a lemma which intuitively says that if the operation on the key x goes on for at least l steps, then there are either "many" keys hashing to the interval $h(x) + [l]$, or there are "many" keys that hash to some interval having $h(x)$ as its right endpoint.

LEMMA 4.1. *For any $l > 0$ and $\bar{\alpha} \in (0,1)$, if $h(x) + [l] \subseteq P$ then at least one of the following holds:*

1. $\left|\{y \in S \setminus \{x\} \ : \ h(y) \in (h(x) + [l])\}\right| \geq \frac{1+\bar{\alpha}}{2} l - 1$, *or*
2. $(\exists \ell) \ \left|\{y \in S \ : \ h(y) \in [h(x) - \ell, \, h(x))\}\right| \geq \ell + \frac{1-\bar{\alpha}}{2} l$.

Proof. Suppose that $\left|\{y \in S \setminus \{x\} \ : \ h(y) \in (h(x) + [l])\}\right| < \frac{1+\bar{\alpha}}{2} l - 1$. Then in either case, $x \in S$ or $x \notin S$, it holds that $\left|\{y \in S \ : \ h(y) \in (h(x) + [l])\}\right| < \frac{1+\bar{\alpha}}{2} l$. Let

$$l' \stackrel{\text{def}}{=} \max\{\ell \ : \ [h(x) - \ell, \, h(x)] \subseteq P\} \ .$$

Now, fix any way of placing the keys in the hash table, e.g., suppose that keys are inserted in sorted order. Consider the set $S^* \subseteq S$ of keys stored in the interval $I = [h(x) - l', \, h(x) + l - 1]$. By the choice of l' there must be an empty position to the left of I, so $h(S^*) \subseteq I$. This means:

$$\begin{aligned}
\left|\{y \in S \ : \ h(y) \in [h(x) - l', \, h(x))\}\right| &\geq \left|\{y \in S^* \ : \ h(y) \in [h(x) - l', \, h(x))\}\right| \\
&\geq |S^*| - \left|\{y \in S^* \ : \ h(y) \in (h(x) + [l])\}\right| \\
&> |I| - \tfrac{1+\bar{\alpha}}{2} l \\
&= l' + \tfrac{1-\bar{\alpha}}{2} l \ .
\end{aligned}$$

□

The next lemma upper bounds the probability that there exists some interval of form $[h(x) - \ell, \, h(x))$ having $\ell + d$ keys hashing into it, with d being a parameter. The derived bound will later be used to cover the case 2 from Lemma 4.1.

LEMMA 4.2. *Let $S \subseteq U$ be a set of size n, and $\mathcal{H}$ a 5-wise independent family of functions from U to R with a maximum load of $\bar{\alpha} < 1$. If h is chosen uniformly at*

random from $\mathcal{H}$, then for any $x \in U$ and $\lambda > 0$,

$$\Pr\left\{\max_{\ell}\left(\left|\{y \in S \,:\, h(y) \in [h(x)-\ell,\, h(x))\}\right| - \ell\right) \geq \frac{\lambda+1}{(1-\bar{\alpha})^{3/2}}\right\} < \frac{8\bar{\alpha}}{\lambda^2} \ .$$

Proof. We will use the symbol Δ to denote $\lceil \frac{\lambda}{2}(1-\bar{\alpha})^{-3/2} \rceil$. Let A_i be the event that

$$\left|\{y \in S \,:\, h(y) \in [h(x)-i\Delta,\, h(x))\}\right| - i\Delta \geq \Delta \ .$$

We claim that it is sufficient to show $\Pr\left(\bigcup_{i>0} A_i\right) < \frac{8\bar{\alpha}}{\lambda^2}$. To see this, suppose that $\left|\{y \in S \,:\, h(y) \in [h(x)-\ell,\, h(x))\}\right| - \ell \geq \frac{\lambda+1}{(1-\bar{\alpha})^{3/2}}$, for some ℓ. Let $i' = \lceil \frac{\ell}{\Delta} \rceil$. Then

$$\begin{aligned}\left|\{y \in S \,:\, h(y) \in [h(x)-i'\Delta,\, h(x))\}\right| &\geq \ell + \frac{\lambda+1}{(1-\bar{\alpha})^{3/2}} \\ &\geq i'\Delta - (\Delta-1) + \lambda(1-\bar{\alpha})^{-3/2} + 1 \\ &\geq i'\Delta + \frac{\lambda}{2}(1-\bar{\alpha})^{-3/2} + 1 \geq i'\Delta + \Delta \ .\end{aligned}$$

In this lemma we use a simple upper bound $\Pr(\bigcup_{i>0} A_i) \leq \sum_{i>0} \Pr(A_i)$. We use Lemma 2.1 to estimate each value $\Pr(A_i)$. Note that intersections of any interval $[h(x)-\ell,\, h(x))$ with the sets $h(S \setminus \{x\})$ and $h(S)$ are the same.

$$\sum_{i>0} \Pr(A_i) \leq \sum_{i>0} \frac{4\bar{\alpha}(i\Delta)^2}{((1-\bar{\alpha})i\Delta + \Delta)^4} \leq \frac{4\bar{\alpha}}{\Delta^2} \sum_{t} \frac{(\frac{t}{1-\bar{\alpha}})^2}{(t+1)^4}$$

We used the substitution $t = (1-\bar{\alpha})i$. The last sum is over $t \in \{1-\bar{\alpha},\ 2(1-\bar{\alpha}), \ldots\}$. The function $\frac{t^2}{(1+t)^4}$ is first increasing and then decreasing on $[0,\infty)$. Thus the sum can be bounded by the integral $\frac{1}{1-\bar{\alpha}} \int_{1-\bar{\alpha}}^{\infty} \frac{t^2}{(1+t)^4} dt$ plus the value of the biggest term in the sum.

$$\begin{aligned}\sum_{i>0} \Pr(A_i) &< \frac{4\bar{\alpha}}{\Delta^2} \frac{1}{(1-\bar{\alpha})^2} \left(\max_{t>0} \frac{t^2}{(1+t)^4} + \frac{1}{1-\bar{\alpha}} \int_{1-\bar{\alpha}}^{\infty} \frac{t^2}{(1+t)^4} dt \right) \\ &< \frac{4\bar{\alpha}}{\Delta^2} \frac{1}{(1-\bar{\alpha})^3} \left(\frac{1}{10} + \int_0^{\infty} \frac{t^2}{(1+t)^4} dt \right) \\ &< \frac{2\bar{\alpha}}{\Delta^2}(1-\bar{\alpha})^{-3} \leq \frac{8\bar{\alpha}}{\lambda^2} \ .\end{aligned}$$

□

THEOREM 4.3. *Consider any sequence of operations (insertions, deletions, and lookups) in a linear probing hash table where the hash function h used has been chosen uniformly at random from a 5-wise independent family of functions $\mathcal{H}$. Let n and $\bar{\alpha} < 1$ denote, respectively, the maximum number of keys in the table during a particular operation and the corresponding maximum load. Then the expected number of probes made during that operation is $O((1-\bar{\alpha})^{-5/2})$.*

Proof. We refer to x, S, and P as defined previously in this section. As argued above, the expected probe count is bounded by

$$1 + \sum_{l>0} \Pr\{h(x) + [l] \subseteq P\} \ .$$

Let $l_0 = \frac{10}{(1-\bar{\alpha})^{5/2}}$. For $l \le l_0$ we use the trivial upper bound $\Pr\{h(x) + [l] \subseteq P\} \le 1$. In the following we consider the case $l > l_0$.

Let A_l be the event that $\big|\{y \in S \setminus \{x\} \,:\, h(y) \in (h(x) + [l])\}\big| \ge \frac{1+\bar{\alpha}}{2} l - 1$, and let B_l be the event that $(\exists \ell)\ \big|\{y \in S \,:\, h(y) \in [h(x) - \ell,\, h(x))\}\big| \ge \ell + \frac{1-\bar{\alpha}}{2} l$. Lemma 4.1 implies that

$$\sum_{l>l_0} \Pr\{h(x) + [l] \subseteq P\} \le \sum_{l>l_0} \big(\Pr(A_l) + \Pr(B_l)\big) \ .$$

Estimates of $\Pr(A_l)$ and $\Pr(B_l)$ are obtained from Lemma 2.1 and Lemma 4.2 respectively:

$$\begin{aligned}
\sum_{l>l_0} \big(\Pr(A_l) + \Pr(B_l)\big) &< \sum_{l>l_0} \left(\frac{4\bar{\alpha} l^2}{(\frac{1-\bar{\alpha}}{2} l - 1)^4} + \frac{8\bar{\alpha}}{(\frac{(1-\bar{\alpha})^{5/2}}{2} l - 1)^2} \right) \\
&= O\Big(\bar{\alpha} \sum_{l>l_0} \Big((1-\bar{\alpha})^{-4} l^{-2} + (1-\bar{\alpha})^{-5} l^{-2} \Big) \Big) \\
&= O\Big((1-\bar{\alpha})^{-5} / l_0 \Big) = O((1-\bar{\alpha})^{-5/2}) \ .
\end{aligned}$$

□

5. Open problems. This paper and its followup work have largely settled the question of practical and provably good hash functions for linear probing. Challenges remain with respect to finding such hash functions for a number of other important hashing methods. Very recently, Pătraşcu and Thorup [13] showed that static cuckoo hashing [10] works well with the simplest tabulation-based hashing method. Still, the problem of finding such hash functions with logarithmic description size remains open. Also, if we consider the problem of hashing a set into $n/\log n$ buckets such that the number of keys in each bucket is $O(\log n)$ with high probability, there is no known explicit class achieving this with function descriptions of $O(\log |U|)$ bits. Possibly, such families could be designed using efficient circuits, rather than a standard CPU instruction set.

Acknowledgments. We thank Martin Dietzfelbinger and an anonymous reviewer for numerous suggestions improving the presentation of the original paper [9]. In particular, we thank Martin Dietzfelbinger for showing us a simplified proof of Lemma 2.1. Finally, we thank Mihai Pătraşcu and Mikkel Thorup for useful discussions of their recent work on linear probing.

REFERENCES

[1] J. R. Black, C. U. Martel, and H. Qi, *Graph and hashing algorithms for modern architectures: Design and performance*, in Algorithm Engineering, 2nd International Workshop, WAE '92, Saarbrücken, Germany, August 20-22, 1998, Proceedings, K. Mehlhorn, ed., Max-Planck-Institut für Informatik, 1998, pp. 37–48.

[2] G. E. Blelloch and D. Golovin, *Strongly history-independent hashing with applications*, in Proceedings of Foundations of Computer Science (FOCS '07), IEEE Computer Society, 2007, pp. 272–282.

[3] J. L. Carter and M. N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[4] M. Dietzfelbinger and C. Weidling, *Balanced allocation and dictionaries with tightly packed constant size bins*, in ICALP, vol. 3580 of Lecture Notes in Computer Science, Springer, 2005, pp. 166–178.

[5] G. L. Heileman and W. Luo., *How caching affects hashing*, in Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX05), SIAM, 2005, pp. 141–154.
[6] D. E. Knuth, *Notes on "open" addressing*, July 22 1963. Unpublished memorandum. Available at `ftp://ftp.inria.fr/INRIA/Projects/algo/web/AofA/Research/src/first.ps.gz`.
[7] D. E. Knuth, *Sorting and Searching*, vol. 3 of The Art of Computer Programming, Addison-Wesley Publishing Co., Reading, Mass., second ed., 1998.
[8] C. P. Kruskal, L. Rudolph, and M. Snir, *A complexity theory of efficient parallel algorithms*, Theoretical Computer Science, 71 (1990), pp. 95–132.
[9] A. Pagh, R. Pagh, and M. Ružić, *Linear probing with constant independence*, SIAM Journal on Computing, 39 (2009), pp. 1107–1120.
[10] R. Pagh and F. F. Rodler, *Cuckoo hashing*, Journal of Algorithms, 51 (2004), pp. 122–144.
[11] H. Prodinger and W. S. (eds.), *Special issue on average case analysis of algorithms*, Algorithmica, 22 (1998). Preface.
[12] M. Pătraşcu and M. Thorup, *On the k -independence required by linear probing and minwise independence*, in Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP '10), vol. 6198 of Lecture Notes in Computer Science, Springer, 2010, pp. 715–726.
[13] ———, *The power of simple tabulation hashing*, in Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC '11), ACM Press, 2011. To appear.
[14] J. P. Schmidt and A. Siegel, *The analysis of closed hashing under limited randomness (extended abstract)*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90), ACM Press, 1990, pp. 224–234.
[15] A. Siegel, *On universal classes of extremely random constant-time hash functions*, SIAM J. Comput., 33 (2004), pp. 505–543.
[16] A. Siegel and J. Schmidt, *Closed hashing is computable and optimally randomizable with universal hash functions*, Technical Report TR1995-687, New York University, Apr., 1995.
[17] M. Thorup, *String hashing for linear probing*, in Proceedings of the 20th Annual Symposium on Discrete Algorithms (SODA '09), SIAM, 2009, pp. 655–664.
[18] M. Thorup and Y. Zhang, *Tabulation based 4-universal hashing with applications to second moment estimation.*, in Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '04), 2004, pp. 615–624.
[19] M. N. Wegman and J. L. Carter, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265–279.
[20] A. Wigderson, *The amazing power of pairwise independence*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC '94), 1994, pp. 645–647.