UNIFORM HASHING IN CONSTANT TIME AND OPTIMAL SPACE[*]

ANNA PAGH[†] AND RASMUS PAGH[‡]

Abstract. Many algorithms and data structures employing hashing have been analyzed under the *uniform hashing* assumption, i.e., the assumption that hash functions behave like truly random functions. Starting with the discovery of universal hash functions, many researchers have studied to what extent this theoretical ideal can be realized by hash functions that do not take up too much space and can be evaluated quickly. In this paper we present an almost ideal solution to this problem: A hash function $h : U \to V$ that, on any set of n inputs, behaves like a truly random function with high probability, can be evaluated in constant time on a RAM, and can be stored in $(1+\epsilon)n \lg |V| + O(n + \lg\lg |U|)$ bits. Here ϵ can be chosen to be any positive constant, so this essentially matches the entropy lower bound. For many hashing schemes this is the first hash function that makes their uniform hashing analysis come true, with high probability, without incurring overhead in time or space.

Key words. Hash function, uniform hashing, randomized algorithms

AMS subject classifications. 68P5, 68P10, 68P20

1. Introduction.

Hashing is an important tool in randomized algorithms and data structures, with applications in such diverse fields as information retrieval, complexity theory, data mining, cryptology, and parallel algorithms. Many algorithms using hashing have been analyzed under the assumption of *uniform hashing*, i.e., the idealized assumption that the hash function employed is a truly random function. In this paper we present a theoretical justification for such analyses, in the form of the construction of a hash function that makes the uniform hashing assumption "come true" with high probability. Our hash function can be evaluated in constant time on a RAM, and its description uses very close to the minimum possible space.

1.1. History.

According to Knuth [16], the idea of hashing was originated in 1953 by H. P. Luhn. The basic idea is to use a function $h : U \to V$, called a *hash function*, that "mimics" a random function. In this way a "random" value $h(x)$ can be associated with each element from the domain U. In this paper, as in most other hash function constructions, we consider a universe of the form $U = \{0, \ldots, u-1\}$ and a range of the form $V = \{0, \ldots, v-1\}$, where $1 < v \leq u$.

Representation of a random function requires $u \lg v$ bits, so it is usually not feasible to actually store a randomly chosen function. For many years hashing was largely a heuristic, and practitioners used fixed functions that were empirically found to work well in cases where uniform hashing could be shown to work well.

The gap between hashing algorithms and their analysis narrowed with the advent of *universal hashing* [6]. The key insight was that it is often possible to get provable performance guarantees by choosing hash functions at random from a small family of functions (rather than from the family of all functions). The importance of the family being small is, of course, that a function from the family can be stored succinctly.

Hash functions are usually chosen uniformly at random from some *family* $\mathcal{H}$ of

[*] A preliminary version of this paper appeared at Symposium on Theory of Computation (STOC), 2003.

[†] Anoto Group AB, Emdalavägen 18, 223 69 Lund, Sweden. Work done at BRICS, Department of Computer Science, University of Aarhus, Denmark, and at IT University of Copenhagen, Denmark.

[‡] IT University of Copenhagen, Rued Langgaards Vej 7, 2300 København S, Denmark. Work done in part at BRICS, Department of Computer Science, University of Aarhus, Denmark.

hash functions. For example, the family

$$\{x \mapsto ((ax+b) \bmod p) \bmod v \mid 0 < a < p,\, 0 \leq b < p\},$$

first studied by Carter and Wegman [6], has many known applications. The family is described by the parameters p and v, while a particular function in the family is given by the values of parameters a and b. In our results, we will distinguish between the space needed to represent the family and the space needed to represent a function in the family.

One property of the choice of hash function that often suffices to give performance guarantees is that it maps each set of k elements in U to uniformly random and independent values, where k is some parameter that depends on the application. If this holds for a random function from a family $\mathcal{H}$, we say that $\mathcal{H}$ is *k-wise independent*. There exist such function families whose functions can be stored in $O(k \lg u)$ bits of space [25]. For many years, all known k-wise independent families with nontrivial space usage required time $\Omega(k)$ for evaluating a hash function. A breakthrough was made by Siegel [23], who showed that high independence is achievable with relatively small families of hash functions that can be evaluated in *constant* time on a RAM.

The *RAM model* used in Siegel's result, as well as in this paper, is a standard unit cost RAM with an instruction set that includes multiplication, and a word size of $\Theta(\lg u)$ bits. The RAM has access to a source of random bits, and in particular we assume that a random value in V and a random word can be generated in constant time.

The two main performance parameters of a hash function family is the space needed to represent a function and the time necessary to compute a given function value from a representation. A tight bound on the number of bits needed to achieve k-wise independence is $\Theta(k \lg u)$ bits [3, 7]. Sometimes there is a trade-off between the space used to represent a function and its evaluation time. For example, Siegel [23] shows that if $u = k^{1+\Omega(1)}$, it is necessary to use $k^{1+\Omega(1)} \lg v$ bits of space to achieve constant evaluation time.

Siegel's construction of a k-wise independent family comes close to this lower bound (see Theorem 2.3). If one applies this family with $k = n$ to a set S of n elements, it will map these to independent and uniformly random values. We say that it is *uniform on S*. However, the space usage is superlinear meaning that, in many possible applications, the hash function description itself becomes asymptotically larger than all other parts of the data structure.

1.2. Our result. In this paper we present a family of hash functions that has the same performance as Siegel's family on any *particular* set of n elements, and space usage close to the lower bound of $n \lg v + \lg \lg_v u$ bits shown in Section 5. The previously best construction using $O(n \lg v + \lg \lg u)$ space is based on evaluation of a degree $n-1$ polynomial over a finite field, and has $\Omega(n)$ evaluation time.

THEOREM 1.1. *Let $S \subseteq U = \{0, \ldots, u-1\}$ be a set of $n > 1$ elements. For any constants $c > 0$ and $\epsilon > 0$, and for $1 < v < u$, there is a RAM algorithm that, using time $\lg n (\lg v)^{O(1)}$ and $O(\lg n + \lg \lg u)$ bits of space, selects a family $\mathcal{H}$ of functions from U to $V = \{0, \ldots, v-1\}$ (independent of S) such that:*

- *$\mathcal{H}$ is k-wise independent when restricted to S, with probability $1 - O(\frac{1}{n^c})$.*
- *A function in $\mathcal{H}$ can be represented by a RAM data structure using space $(1+\epsilon) n \lg v + O(n)$ bits such that function values can be computed in constant time. The data structure of a random function in $\mathcal{H}$ can be constructed in time $O(n)$.*

As our hash functions are optimal with respect to evaluation time and essentially optimal with respect to space usage, the only possible significant improvement would be an error probability that decreases more rapidly with n. Such a result could possibly be achieved for $u = n^{O(1)}$ by explicit constructions of certain expanders in Siegel's hash function construction.

Techniques. Our main technical ingredient is to use the two-choice paradigm [4] that has recently found a number of applications in load balancing and data structures. The central fact we make use of is that if we associate, using hashing, two memory locations in a linear size array with every element of the set S, then there exists (whp.) a way of associating keys with unique memory locations [19]. Essentially, each key gets the independence of its hash value from random bits at the memory location with which it is associated. A complication is that one needs to take care of cyclic dependencies that may arise, but this involves only a logarithmic number of elements, whp. The solution is to add a hash function that is independent (whp.) on the set of problematic elements.

Perspective. It should be noted that a data structure with slightly different functionality is very easy to construct: Use a high performance dictionary such as the one in [9] to store elements from U and associated "hash values" from V. When a new function value is needed, it is randomly generated "on the fly" and stored with the element in the dictionary. The space needed for this is $O(n(\lg u + \lg v))$ bits, which is a constant factor from the space usage achieved by our data structure if $u = v^{O(1)}$. The main difference between this and the data structure of Theorem 1.1 is that our hash function can be generated once and for all, after which it may be used without any need for random bits. This also means that our hash function may be distributed and used by many parties without any need for additional communication. Another distinguishing feature is that our hash function will be uniform with high probability on *each* of $n^{O(1)}$ sets of size n. Thus it may be shared among many independently running algorithms or data structures.

1.3. Implications. The fact that the space usage of our hash function is linear in n means that a large class of hashing schemes can be implemented to perform, with high probability, *exactly as if uniform hashing was used*, increasing space by at most a constant factor. This means that our family makes a large number of analyses performed under the uniform hashing assumption "come true" with high probability.

Two comprehensive surveys of early data structures analyzed under the uniform hashing assumption can be found in the monographs of Gonnet [13] and Knuth [16]. Gonnet provides more than 100 references to books, surveys and papers dealing with the analysis of classic hashing algorithms. This large body of work has made the characteristics of these schemes very well understood, under the uniform hashing assumption. As the classic hashing algorithms are often very simple to implement, and efficient in practice, they seem to be more commonly used in practice than newer schemes with provably good behavior. While our family may not be of practical importance for these hashing schemes, it does provide a theoretical "bridge" justifying the uniform hashing assumption for a large class of them. Previously, such justifications have been made for much more narrow classes of hashing schemes, and have only dealt with certain performance parameters (see, e.g., [20, 21]). More details on applications of our scheme in hashing data structures can be found in the conference version of this paper [17].

In addition to the classic hashing schemes, our result provides the first provably efficient hashing based implementation of load balancing schemes of Azar et al. [4] and

Vöcking [24]. The fact that hash functions can be used to perform the random choices in these algorithms means that it is possible to retrace any previous load balancing decision by checking a small number of possible choices.

Finally, our construction has an application in cryptography, where it "derandomizes" a construction of Bellare et al. [5] for the most important parameters (see discussion in Section 3.1.1).

1.4. Overview of the paper. The organization of the paper is as follows. In Section 2 we provide the background information necessary to understand our construction. Specifically, we survey Siegel's construction, which will play an important role. Section 3 presents our initial construction, which achieves space that is within a constant factor of optimal. Section 4 shows, by a general reduction, how to reduce the space to (essentially) $1+\epsilon$ times the space lower bound shown in Section 5.

2. Background.

Theorem 1.1 can be seen as an improvement of Siegel's family of high performance hash functions [23]. The motivation for Siegel's work was that many algorithms employing hashing can be shown to work well if the hash functions are chosen at random from a *k-wise independent* family of functions, for suitably large k.

DEFINITION 2.1. *A family $\mathcal{H}$ of functions from U to V is* k-wise independent *if, for any distinct elements $x_1,\ldots,x_k \in U$, and any $y_1,\ldots,y_k \in V$, when $h \in \mathcal{H}$ is chosen uniformly at random,*

$$\Pr[h(x_1)=y_1 \wedge \cdots \wedge h(x_k)=y_k] = |V|^{-k} \ .$$

In other words, a random function from a k-wise independent family acts like a truly random function on any set of k elements of U. We note that several relaxed notions of k-wise independence exist, e.g. the notion of (c,k)-universality [8]. However, we don't know of any data structures in the literature that allow evaluation of a function from a (c,k)-universal (or similar) family in time $o(k)$, except for those achieving k-wise independence.

Siegel's construction. Siegel showed that for arbitrary constants $c>0$ and $\epsilon>0$ it is possible to construct, using $O(u^{\sqrt{\lg k/\lg u}+\epsilon}\lg v)$ bits of space, a family of functions from U to V with the following properties:

- It is k-wise independent with probability at least $1-u^{-c}$.
- There is a RAM data structure of $O(u^{\sqrt{\lg k/\lg u}+\epsilon}\lg v)$ bits representing its functions such that function values can be computed in constant time.

Siegel mainly considered the case $k=u^{o(1)}$, e.g., $k=O(\lg u)$, where the space usage is dominated by the u^ϵ term. The positive probability that the family is not k-wise independent is due to the fact that Siegel's construction relies on a certain type of expander graph that, in lack of an explicit construction, is found by selecting a graph at random (and storing it). However, there is a small chance that the randomly chosen graph is not the desired expander, in which case there is no guarantee on the performance of the family. Also, there seems to be no known efficient way of generating a graph at random and verifying that it is the desired expander. (However, a slightly different class of expanders can be efficiently generated in this way [2].)

Space lower bound. It is inevitable that the space usage grows with u when constant evaluation time is required. Siegel shows the following trade-off between evaluation time and the size of the data structure:

THEOREM 2.2. (Siegel [23]) *Consider any k-wise independent family $\mathcal{H}$ of functions from U to V, and any RAM data structure using m words of $O(\lg v)$ bits to* *represent a function from $\mathcal{H}$. Then there exists $h \in \mathcal{H}$ and $x \in U$ such that the data structure for h requires time $\Omega(\min(\lg_{m/k}(u/k), k))$ on input x.*

Note that when using optimal space, i.e., $m = O(k)$, the time needed to evaluate a function is $\Omega(\min(\lg(u/k), k))$.

Applying domain reduction. Theorem 2.2 establishes that high independence requires either high evaluation time or high space usage when u is large. A standard way of getting around problems with hashing from a large domain is to first perform a *domain reduction*, where elements of U are mapped to elements of a smaller domain U' using universal hashing. As this mapping cannot be 1-1, the domain reduction forces some hash function values to be identical. However, for any particular set S of n elements, the probability of two elements in S mapping to the same element of U' is $O(n^{-c})$ if $|U'| \geq n^{c+2}$. One universal family, suggested implicitly in [12] and explicitly in [18], uses primes in a certain range. A function from the family can be generated in expected time $(\lg |U'| + \lg\lg u)^{O(1)}$ and stored in $O(\lg |U'| + \lg\lg u)$ bits. Another universal family [8] has functions that can be generated in constant time and stored in $O(\lg u)$ bits. Both families support constant time evaluation of functions. In the following we will state all results using the former universal family, obtaining the smallest possible space at the cost of a modest amount of precomputation.

Using domain reduction with Siegel's family described above, one gets the following result. For $k = n$ it is similar to our main theorem, the main difference being that the space usage is superlinear.

THEOREM 2.3. (Siegel [23]) *Let $S \subseteq U = \{0, \ldots, u-1\}$ be a set of n elements. For any constants $\epsilon > 0$ and $c > 0$ there is a RAM algorithm constructing a random family $\mathcal{SI}(U, V, k, n, c, \epsilon)$ of functions from U to $V = \{0, \ldots, v-1\}$ in expected time $O(s) + (\lg\lg u)^{O(1)}$ and $O(s \lg v + \lg\lg u)$ bits of space, where $s = n^{\sqrt{(c+2)\lg k/\lg n + \epsilon}}$, such that:*

- *With probability $1 - O(\frac{1}{n^c})$ the family is k-wise independent when restricted to S.*
- *There is a RAM data structure of $O(s \lg v + \lg\lg u)$ bits representing its functions such that function values can be computed in constant time. The data structure can be initialized to a random function in time $O(s)$.*

Notice that the space usage is $\omega(k^{\sqrt{2}})$ bits for all parameters, so it is truly superlinear in k. With currently known ways of constructing expanders, Siegel's hash function family exhibits high constant factors.

Other constructions. Other proposals for high performance hash functions, due to Dietzfelbinger and Meyer auf der Heide [9, 10], appear more practical than Siegel's. However, these families only exhibit $O(1)$-wise independence and are difficult to analyze in general.

Dietzfelbinger and Woelfel [11] have analyzed a family similar to the abovementioned high performance hash functions, and shown that it may be used for hashing schemes whose analysis rests on a bipartite graph defined by a pair of hash functions. In particular, they are able to give an alternative to our initial uniform hashing construction, described in Section 3, that is likely to be more practical. However, their construction restricts the size v of the range to be a prime number.

3. Initial hash function construction.

In this section we describe a hash function family with properties as stated in Theorem 1.1 for *some* constant $\epsilon > 0$. In Section 4 we will extend this to show Theorem 1.1 (where ϵ can be any positive constant). We use the notation $T[i]$ to denote the ith entry in an array T. By $[m]$ we denote the set $\{0, \ldots, m-1\}$.

3.1. The hash function family.

We start with a definition.

Definition 3.1. *Let $\mathcal{G}$ be a family of functions from U to V, and consider functions $i_1, i_2 : U \to [m]$. We define the family of functions*

$$\mathcal{H}(i_1, i_2, \mathcal{G}) = \{x \mapsto (T_1[i_1(x)] + T_2[i_2(x)] + g(x)) \bmod v \mid T_1, T_2 \in V^m \text{ and } g \in \mathcal{G}\}.$$

The hash function family considered in this section uses Siegel's construction of function families to get the functions i_1 and i_2, as well as the family $\mathcal{G}$ in the above definition.

Definition 3.2. *For $n \leq u$ and any constant $c > 0$ we define the random family $\mathcal{H}_{n,c} = \mathcal{H}(i_1, i_2, \mathcal{G})$ of functions as follows: Let $k = \lceil n^{1/(2c+4)} \rceil$ and construct the random families*

$$\mathcal{G} = \mathcal{SI}(U, V, k, n, c, 1/4), \text{ and}$$
$$\mathcal{F} = \mathcal{SI}(U, [4n], k, n, c, 1/4)$$

according to Theorem 2.3. Pick i_1 and i_2 independently at random from $\mathcal{F}$.

3.1.1. Related constructions.

A similar way of constructing a function family was presented in [9]. The essential change in the above definition compared to [9] is that we look up *two* values in tables, rather than just one.

The technique of using multiple lookups in a random (or pseudorandom) table to produce a new random value has previously been used in cryptography by Bellare et al. [5] in connection with stateless evaluation of pseudorandom functions. The construction given in this paper is strictly more general that the one in [5] as we get a random *function* rather than just a way of generating random values. Also, function evaluation is deterministic, whereas the generation procedure in [5] is randomized. Our analysis is completely different from the one in [5].

On the other hand, our analysis shares some features with the analysis of cuckoo hashing in [19], as it rests on the analysis of the same random bipartite graph (generated by Siegel's hash functions). In fact, Dietzfelbinger and Woelfel show in [11] how to base uniform hashing (with range of size that is a prime number) on any hash function that works with cuckoo hashing.

3.2. Properties of the family.

For two functions $i_1, i_2 : U \to [m]$ and a set $S \subseteq U$, let $G(i_1, i_2, S) = (A, B, E)$ be the bipartite graph that has left vertex set $A = \{a_1, \ldots, a_m\}$, right vertex set $B = \{b_1, \ldots, b_m\}$, and edge set

$$E = \{e_x \mid x \in S\}, \text{ where } e_x = (a_{i_1(x)}, b_{i_2(x)}) \ .$$

We consider the edge e_x to be labeled by x. Note that there may be parallel edges with different labels.

We define a *leafless subgraph* $E' \subseteq E$ of a graph as a subset of the edges such that there is no vertex incident to exactly one edge in E'. A graph's *leafless part* $C \subseteq E$ consists of the edges that are on a cycle and the edges that are on a path connecting two cycles. (This is also known as the *2-core*.)

Lemma 3.3. *Let $S \subseteq U$ be a set of n elements and let $\mathcal{G}$ be a family of functions from U to V that is k-wise independent on S. If the total number of edges in the leafless part of $G(i_1, i_2, S) = (A, B, E)$ is at most k, then $\mathcal{H}(i_1, i_2, \mathcal{G})$ is uniform when restricted to S.*

Proof. Let S' be the set of all elements $x \in S$ where the edge e_x with label x is in the leafless part C of $G(i_1, i_2, S)$.

The proof is by induction on $|E \setminus C|$. In the base case we assume that $|E \setminus C| = 0$, i.e., that $S' = S$. Since $|S| \leq k$ it holds for any function h that the function family $x \mapsto (h(x) + g(x)) \bmod v$, where g is chosen from a k-wise independent family, is uniform on S. In particular, $\mathcal{H}(i_1, i_2, \mathcal{G})$ has this property.

For the inductive step, note that among the edges in $E \setminus C$ there has to be at least one edge with one unique endpoint. Let $e_{x^*} = (a_{i_1(x^*)}, b_{i_2(x^*)})$ be such an edge, $x^* \in S \setminus S'$. By symmetry we may assume that $a_{i_1(x^*)}$ is the unique endpoint. The induction hypothesis says that $\mathcal{H}(i_1, i_2, \mathcal{G})$ is uniform on $S \setminus \{x^*\}$, and for $h \in \mathcal{H}(i_1, i_2, \mathcal{G})$ chosen at random, all values $h(x)$ for $x \in S \setminus \{x^*\}$ are independent of the value $T_1[i_1(x^*)]$. These facts mean that, given $g \in \mathcal{G}$ and all entries in vectors T_1 and T_2 except $T_1[i_1(x^*)]$, $h(x^*)$ is uniformly distributed when choosing $T_1[i_1(x^*)]$ at random. Hence $\mathcal{H}(i_1, i_2, \mathcal{G})$ is uniform on S. □

LEMMA 3.4. *For each set S of size n, and for $i_1, i_2 : U \to [4n]$ chosen at random from a family that is k-wise independent on S, $k \geq 32$, the probability that the leafless part C of the graph $G(i_1, i_2, S)$ has size at least k is $n/2^{\Omega(k)}$.*

Proof. Assume that $|C| \geq k$ and that $k \leq n$ is even (this may be assumed without loss of generality). Either there is a *connected* leafless subgraph in $G(i_1, i_2, S)$ of size at least $k/2$ or there is a leafless subgraph of size k', where $k/2 < k' \leq k$. In the first case there is a connected subgraph in $G(i_1, i_2, S)$ with exactly $k/2$ edges and at most $k/2 + 1$ vertices. In the second case there is a subgraph with k' edges and at most k' vertices in $G(i_1, i_2, S)$.

In the following we will count the number of different edge labeled subgraphs with k' edges and at most $k' + 1$ vertices for $k/2 \leq k' \leq k$ to bound the probability of such a subgraph to appear in $G(i_1, i_2, S)$. Hence, we also get an upper bound on the probability that $|C|$ is at least k. Note that since i_1 and i_2 are chosen from a k-wise independent family, each subset of at most k elements of S will map to to random and independent edges. We will only consider subgraphs corresponding to at most k elements of S.

To count the number of different subgraphs with k' edges and at most $k' + 1$ vertices, for $k/2 \leq k' \leq k$, in a bipartite graph $G = (A, B, E)$ with $|A| = |B| = 4n$ and $|E| = n$, we count the number of ways to choose the edge labels, the vertices, and the endpoints of the edges such that they are among the chosen vertices. The k' edge labels can be chosen in $\binom{n}{k'} \leq (en/k')^{k'}$ ways. Since the number of vertices in the subgraph is at most $k' + 1$, and they are chosen from $8n$ vertices in G, the total number of ways in which they can be chosen is bounded by

$$\sum_{i=1}^{k'+1} \binom{8n}{i} \leq (8en/(k'+1))^{k'+1} \ .$$

Let k_a and k_b be the number of vertices chosen from A and B, respectively. The number of ways to choose an edge such that it has both its endpoints among the chosen vertices is $k_a k_b \leq ((k'+1)/2)^{2k'}$. In total, the number of different subgraphs with k' edges and up to $k' + 1$ vertices is at most

$$\begin{aligned}
&(en/k')^{k'} \cdot (8en/(k'+1))^{k'+1} \cdot ((k'+1)/2)^{2k'} \\
&= \tfrac{8en}{k'+1} \cdot (2e^2 \cdot n^2 \cdot \tfrac{k'+1}{k'})^{k'} \\
&\leq \tfrac{8en}{k'+1} \cdot (\tfrac{63}{4} \cdot n^2)^{k'},
\end{aligned}$$

using $k' \geq k/2 \geq 16$.

There are in total $(4n)^{2k'}$ graphs with k' specific edges. In particular, the probability that k' specific edges form a particular graph is $(4n)^{-2k'}$, using k'-wise independence. To get an upper bound on the probability that there is some subgraph with k' edges and at most $k'+1$ vertices, where $k/2 \leq k' \leq k$, we sum over all possible values of k':

$$\begin{aligned}\sum_{k/2\leq k'\leq k} & \tfrac{8en}{k'+1} \cdot (\tfrac{63}{4} \cdot n^2)^{k'} \cdot (4n)^{-2k'} \\ &= \sum_{k/2\leq k'\leq k} \tfrac{8en}{k'+1} \cdot (\tfrac{63}{64})^{k'} \\ &\leq (k/2+1) \cdot \tfrac{8en}{k/2+1} \cdot (\tfrac{63}{64})^{k/2} \\ &= n/2^{\Omega(k)} \ .\end{aligned}$$

□

We now proceed to show Theorem 1.1 for *some* constant $\epsilon > 0$. More precisely, we will show that the random family $\mathcal{H}_{n,c}$ of Definition 3.2 fulfills the requirements in the theorem. The families $\mathcal{G} = \mathcal{SI}(U, V, k, n, c, 1/4)$ and $\mathcal{F} = \mathcal{SI}(U, [4n], k, n, c, 1/4)$ are both k-wise independent with probability $1-n^{-c}$ for sets of size up to n according to Theorem 2.3. If $\mathcal{F}$ is k-wise independent then by Lemma 3.4 the probability that the leafless part of graph $G(i_1, i_2, S)$ has size at most k is at least $1 - n^{-\Omega(k)}$, if $k \geq 32$. We can assume without loss of generality that $k \geq 32$, since otherwise the theorem follows directly from Theorem 2.3. When the leafless part of graph $G(i_1, i_2, S)$ has size at most k then, by Lemma 3.3, $\mathcal{H}_{n,c}$ is uniform on S if $\mathcal{G}$ is k-wise independent. The probability that $\mathcal{G}$ is k-wise independent, $\mathcal{F}$ is k-wise independent, and that the leafless part of the graph $G(i_1, i_2, S)$ has size at most k is altogether $(1-n^{-c})^2(1-2^{-\Omega(k)}) = 1 - O(n^{-c})$.

The construction of $\mathcal{H}_{n,c}$, i.e., constructing $\mathcal{F}$ and $\mathcal{G}$ and choosing i_1 and i_2, can according to Theorem 2.3 be done in expected $O(s) + (\lg\lg u)^{O(1)}$ time and $O(s \lg v)$ bits of space, where $s = n^{\sqrt{(c+2)\lg k/\lg n}+1/4} = O(n^{0.96})$. The space usage of a data structure representing a function from $\mathcal{H}_{n,c}$ is $O(n \lg v)$ bits for T_1 and T_2, and $O(s \lg v + \lg\lg u)$ bits for storing i_1, i_2 and g. The initialization time is dominated by the time used for initializing T_1 and T_2 to random arrays and the $(\lg\lg u)^{O(1)}$ term from Siegel's construction. Function values can clearly be computed in constant time.

4. Achieving near-optimal space. In this section we present a general reduction for decreasing the space used by a uniform hashing construction. Together with Section 3 this will show Theorem 1.1. The idea of the reduction is the following. We construct a data structure of $(1+\epsilon/2)n \lg v + O(n)$ bits representing a function f, such that for any set $S \subseteq U$ of n elements there exists, with probability $1 - O(n^{-c})$, a set $S' \subseteq S$, where $|S'| \geq (1-\epsilon/16)\,n$, such that the following independence property holds:

- The values $(f(x))_{x\in S'}$ are independent and uniform in $[v]$, even when conditioned on particular f-values for elements in $S \backslash S'$.

To be precise, we associate with each possible choice of f a unique set S'. The uniformity property above holds when restricting f to the subset of functions associated with any particular set S'. (The probability space may be empty for some choices of S'.)

Now choose a function h according to a uniform hashing construction for sets of size $(\epsilon/16)\,n$, and error probability $O(n^{-c})$. We consider the function

$$x \mapsto (f(x) + h(x)) \bmod v \ .$$

For a given set $S \subseteq U$ of size n, there exists with probability $1 - O(n^{-c})$ a set S' with properties as stated above. With probability $1 - O(n^{-c})$ the function h then maps the elements of $S \backslash S'$ to uniformly distributed and independent values. Hence, using the independence property of f, the function $x \mapsto (f(x) + h(x)) \bmod v$ must be uniform on the set S with probability $1 - O(n^{-c})$.

Using the uniform hashing construction of Section 3, the space to represent h is $(8\epsilon/16)\,n\lceil \lg v \rceil + o(n) + O(\lg\lg u) = (\epsilon/2)n \lg v + O(n + \lg\lg u)$ bits, so the total space for storing f and h is as required in Theorem 1.1.

4.1. Details of the reduction. It remains to see how to construct f in the desired space bound, with the desired preprocessing time, and such that function values can be computed in constant time. Below we will several times refer to a constant ℓ, which is assumed to be “sufficiently large”. That is, the arguments are valid if ℓ is set to a sufficiently large constant. We first notice that for the case where $v \leq \ell$, Theorem 1.1 was already shown in Section 3. Thus, in the following we may assume that $v > \ell$. Let $p \geq v$ be a prime in the range v to $v + O(v^{2/3})$. We know from [14] that there are $\Omega(v^{2/3}/\lg v)$ such primes, so p can be found in time $\lg n (\lg v)^{O(1)}$ with high probability by sampling [1].

Let $d = \lceil \ell/\epsilon^2 \rceil$, $k = \lceil n^{1/(2c+4)} \rceil$, $r_1 = \lceil (1+\epsilon/2)n/d \rceil$, and let $r_2 > 9d^3/\epsilon$ be a constant. Since v can be assumed to be sufficiently large, we have that $p - v < \epsilon v/(9d)$. Now pick the following functions uniformly at random:

$$\begin{aligned} \rho_1 &: U \to \{0, \ldots, r_1 - 1\} \text{ from } \mathcal{SI}(U, [r_1], k, n, c, 1/4) \\ \rho_2 &: U \to \{0, \ldots, r_2 - 1\} \text{ from } \mathcal{SI}(U, [r_2], k, n, c, 1/4) \end{aligned}$$

Also, pick functions $f_1, \ldots, f_{r_1}$ independently at random from the family of degree $d-1$ polynomials in the field of size p, which is known to be d-wise independent. Note that such a polynomial can be stored in $d\lceil \lg p \rceil = d \lg v + O(d)$ bits, and evaluated in $O(d)$ time using arithmetic modulo p. Without loss of generality we may assume that $\epsilon < 1$ and $p \geq r_2$. Thus we may interpret a value of ρ_2 as an input to $f_1, \ldots, f_{r_1}$, and the following is well-defined:

$$f(x) = f_{\rho_1(x)}(\rho_2(x))$$

Observe that f may have function values up to $p-1$, i.e., not in $[v]$. However, we will define S' such that the function values of elements in S' are uniform in $[v]$, and this suffices for the reduction to work.

4.2. Analysis.

Time and space usage. We first argue that f has the desired properties with respect to storage, preprocessing time, and evaluation time. The storage required for ρ_1 and ρ_2 is bounded by the storage used in the construction of Section 3, so it can be ignored in this context. The functions $f_1, \ldots, f_{r_1}$ require space $r_1(d \lg v + O(d)) = (1+\epsilon/2)n \lg v + O(n)$ bits, and a function can be evaluated in $O(1)$ time, since d is a constant. Selection of ρ_1 and ρ_2 can be done in $o(n)$ time, while construction of $f_1, \ldots, f_{r_1}$ requires expected time $w^{O(1)}$ to find the prime p, and $O(n)$ time to choose the random polynomials.

Independence. To argue that f has the desired independence property, we let $S_i = \{x \in S \mid \rho_1(x) = i\}$ and define S' to be the union of the sets S_i for which:

- $|S_i| \leq d$,
- $|\rho_2(S_i)| = |S_i|$, i.e., ρ_2 is 1-1 on S_i, and
- $f_i(\rho_2(S_i)) \subseteq [v]$

In other words, if we think of ρ_1 as hashing into buckets of capacity d, the set S' consists of those elements that are hashed to a bucket i that does not overflow, whose elements have no collisions under ρ_2, and where f_i produces values in the right range for all elements in the bucket.

Consider a nonempty part of the probability space where f is associated with a particular set S', and has a specific, fixed value on each of the elements in $S \backslash S'$. We will argue that if f is chosen in this part of the probability space, we get a uniform function on S'. First of all, function values of elements hashing to different buckets are completely independent as $f_1, \ldots, f_{r_1}$ are chosen independently. If S_i is part of S', then $|S_i| \leq d$ by definition of S'. Since f_i is d-wise independent and there is no collision among elements in S_i under ρ_2, the f-values of elements in S_i are independent and uniformly random in $[v]$ (because of the requirement $f_i(\rho_2(S_i)) \subseteq [v]$). This concludes the argument for uniformity on S'.

Size of S'. Finally, we need to show that $|S'| \geq (1 - \epsilon/16)\,n$ with probability $1 - O(n^{-c})$. For this we split $[r_1]$ into blocks of at most 2^z consecutive integers $I_j = \{2^z j, \ldots, 2^z(j+1) - 1\}$, $j = 0, 1, 2, \ldots$, where $z = \Omega(\lg n)$ is the highest integer for which $2^z d < k$. If 2^z does not divide r_1 there will be one block of size less than 2^z. If we conservatively assume that all elements of S having a ρ_1-value in this final block will be part of S', it will follow from the arguments below that this will contribute only negligibly to S'. Thus we simply assume that 2^z divides r_1.

First we observe that for all j, $|\cup_{i \in I_j} S_i| < (1-\epsilon/4)2^z d$ with probability $1 - n^{-\omega(1)}$. This follows from Chernoff bounds for random variables with limited dependence [22, Theorem 5.I.b]. On the condition that $| \cup_{i \in I_j} S_i| < (1 - \epsilon/4)2^z d$ (which is assumed in the following) the z least significant bits of the ρ_1-values of elements in $\cup_{i \in I_j} S_i$ will be random and independent for any particular j. We conclude the argument by proving that for any j, $|S' \cap (\cup_{i \in I_j} S_i)| \geq (1 - \epsilon/16)| \cup_{i \in I_j} S_i|$ with probability $1 - O(n^{-\omega(1)})$. By the union bound this will imply that S' has the desired size with probability $1 - O(n^{-\omega(1)})$.

Consider the indicator random variables Y_x, $x \in \cup_{i \in I_j} S_i$, where $Y_x = 1$ if and only if x has the same value under ρ_1 as at least d other elements in $\cup_{i \in I_j} S_i$. Observe that if $Y_x = 1$ then x is not included in S' due to the first requirement in the definition of S. By uniformity of ρ_1 in the z least significant bits, and since the expected number of elements colliding with x is bounded by $(1-\epsilon/4)d$ it follows from classical Chernoff bounds that $\Pr(Y_x = 1) \leq \epsilon/6$. The random variables Y_x are not independent; however, they are *negatively related* [15] which means that we can apply a Chernoff bound on the sum of the Y_xs to show that it is bounded by $(\epsilon/4)| \cup_{i \in I_j} S_i|$ with probability $1 - n^{\omega(1)}$ [15].

Finally consider the indicator random variables X_i, $i \in I_j$ where $X_i = 1$ if and only if $|S_i| \leq d$ and either $|\rho_2(S_i)| < |S_i|$ or $f_i(\rho_2(S_i)) \not\subseteq [v]$. That is, X_i indicates whether the set S_i fails to be included in S' because of at least one of the two last requirements in the definition of S'. For each variable X_i equal to 1 we have at most d elements (those in S_i) that are not part of S'. We next show that with probability $1 - n^{\omega(1)}$ the sum $\sum_{i \in I_j} X_i$ is bounded by $2^z \epsilon/(4d)$, which means that the number of elements not included in S' due to requirements two and three is at most $(\epsilon/4)| \cup_{i \in I_j} S_i|$. Since

ρ_2 is independent on all elements in $\cup_{i\in I_j} S_i$, the X_i are independent. By the choice of p and r_2 we have that for all i, $\Pr(X_i = 1) \le \frac{p-v}{v} + \binom{d}{2}/r_2 < 2\epsilon/(9d)$. Hence by Chernoff bounds $\sum_{i\in I_j} X_i < 2^z\epsilon/(4d)$ with probability $1 - n^{\omega(1)}$. Together with the similar bound above for the first requirement this shows that S' has the desired size with high probability.

The combined construction. For ease of reference we state the full uniform hashing construction used to show Theorem 1:

$$x \mapsto (f_{\rho_1(x)}(\rho_2(x)) + T_1[i_1(x)] + T_2[i_2(x)] + g(x)) \bmod v$$

5. Space lower bound. We now show that our space usage in bits is close to the best possible. To this end, note that any data structure achieving n-wise independence on a set S of n elements, with nonzero probability, must be able to represent every function from S to V.

THEOREM 5.1. *For integers $u \ge n \ge 2$ and $v \ge 2$, let $U = \{0, \ldots, u-1\}$ and $V = \{0, \ldots, v-1\}$. Any data structure representing functions $h : U \to V$ such that the restriction $h|_S$ to any set $S \subseteq U$ of n elements can be an arbitrary function from S to V must use space $\max(n \lg v, \lg \lg_v u)$ bits.*

Proof. Even for fixed S, $n \lg v$ bits are necessary to be able to represent all functions from S to V. Secondly, if the data structure can represent fewer than $\lg_v u$ different functions, there will be elements $x_1, x_2 \in U$ such that all functions map x_1 and x_2 to the same value, contradicting the assumptions of the theorem. Thus the data structure must have at least $\lg \lg_v u$ bits. □

Note that when $\lg v < \lg \lg u$ the second term in the lower bound is $\Omega(\lg \lg u)$, so the lower bound is $\Omega(n \lg v + \lg \lg u)$ bits.

Acknowledgement. We would like to thank Martin Dietzfelbinger for enlightening discussions, and in particular for pointing out an error in the quotation of Siegel's result in the conference version of this paper. Also, we thank an anonymous reviewer for help on improving the presentation.

REFERENCES

[1] MANINDRA AGRAWAL, NEERAJ KAYAL, AND NITIN SAXENA, *PRIMES is in P*, Annals of Mathematics, 160 (2004), pp. 781–793.

[2] NOGA ALON, *Eigenvalues and expanders*, Combinatorica, 6 (1986), pp. 83–96.

[3] NOGA ALON, LÁSZLÓ BABAI, AND ALON ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[4] YOSSI AZAR, ANDREI Z. BRODER, ANNA R. KARLIN, AND ELI UPFAL, *Balanced allocations*, SIAM J. Comput., 29 (1999), pp. 180–200.

[5] MIHIR BELLARE, ODED GOLDREICH, AND HUGO KRAWCZYK, *Stateless evaluation of pseudorandom functions: Security beyond the birthday barrier*, in Proc. of 19th annual international cryptology conference (CRYPTO'99), vol. 1666 of Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 270–287.

[6] J. LAWRENCE CARTER AND MARK N. WEGMAN, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[7] BENNY CHOR, ODED GOLDREICH, JOHAN HASTAD, JOEL FRIEDMAN, STEVEN RUDICH, AND ROMAN SMOLENSKY, *The bit extraction problem of t-resilient functions (preliminary version)*, in Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS '85), IEEE Comput. Soc. Press, 1985, pp. 396–407.

[8] MARTIN DIETZFELBINGER, *Universal hashing and k-wise independent random variables via integer arithmetic without primes*, in Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (STACS '96), vol. 1046 of Lecture Notes in Computer Science, Springer-Verlag, 1996, pp. 569–580.

[9] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide, *A new universal class of hash functions and dynamic hashing in real time*, in Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP '90), vol. 443 of Lecture Notes in Computer Science, Springer-Verlag, 1990, pp. 6–19.

[10] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide, *High performance universal hashing, with applications to shared memory simulations*, in Data structures and efficient algorithms, vol. 594 of Lecture Notes in Computer Science, Springer, 1992, pp. 250–269.

[11] Martin Dietzfelbinger and Philipp Woelfel, *Almost random graphs with simple hash functions*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC '03), 2003, pp. 629–638.

[12] Michael L. Fredman, János Komlós, and Endre Szemerédi, *Storing a sparse table with $O(1)$ worst case access time*, J. Assoc. Comput. Mach., 31 (1984), pp. 538–544.

[13] Gaston Gonnet, *Handbook of Algorithms and Data Structures*, Addison-Wesley Publishing Co., 1984.

[14] David R. Heath-Brown and Henryk Iwaniec, *On the difference between consecutive primes*, Invent. Math., 55 (1979), pp. 49–69.

[15] Svante Janson, *Large deviation inequalities for sums of indicator variables*, Tech. Report 34, Department of Mathematics, Uppsala University, 1993.

[16] Donald E. Knuth, *Sorting and Searching*, vol. 3 of The Art of Computer Programming, Addison-Wesley Publishing Co., Reading, Mass., second ed., 1998.

[17] Anna Östlin and Rasmus Pagh, *Uniform hashing in constant time and linear space*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC '03), ACM Press, 2003, pp. 622–628.

[18] Rasmus Pagh, *Dispersing Hash Functions*, in Proceedings of the 4th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '00), vol. 8 of Proceedings in Informatics, Carleton Scientific, 2000, pp. 53–67.

[19] Rasmus Pagh and Flemming Friche Rodler, *Cuckoo hashing*, Journal of Algorithms, 51 (2004), pp. 122–144.

[20] Jeanette P. Schmidt and Alan Siegel, *On aspects of universality and performance for closed hashing (extended abstract)*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89), ACM Press, 1989, pp. 355–366.

[21] ———, *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.

[22] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan, *Chernoff-Hoeffding bounds for applications with limited independence*, SIAM J. Discrete Math., 8 (1995), pp. 223–250.

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

[24] Berthold Vöcking, *How asymmetry helps load balancing*, J. Assoc. Comput. Mach., 50 (2003), pp. 568–589.

[25] Mark N. Wegman and J. Lawrence Carter, *New hash functions and their use in authentication and set equality*, J. Comput. System Sci., 22 (1981), pp. 265–279.