Simulating Uniform Hashing in Constant Time and Optimal Space

Anna Östlin
Rasmus Pagh

BRICS Report Series RS-02-27

ISSN 0909-0878 2002

Simulating Uniform Hashing in Constant Time and Optimal Space*

Anna Östlin and Rasmus Pagh
BRICS†
Department of Computer Science
University of Aarhus, Denmark
{annao,pagh}@brics.dk

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. In this paper it is shown how to implement hash functions that can be evaluated on a RAM in constant time, and behave like truly random functions on any set of n inputs, with high probability. The space needed to represent a function is $O(n)$ words, which is the best possible (and a polynomial improvement compared to previous fast hash functions). As a consequence, a broad class of hashing schemes can be implemented to meet, with high probability, the performance guarantees of their uniform hashing analysis.

1 Introduction

Hashing is an important tool for designing and implementing randomized algorithms and data structures. 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. This has been useful in many applications in, for example, information retrieval, data mining, cryptology, and parallel algorithms.

Many algorithms have been carefully analyzed under the assumption of *uniform hashing*, i.e., assuming that the hash function employed is a truly random function. As the representation of a random function requires $|U| \log |V|$ bits, it is usually not feasible to actually store a randomly chosen function. For many years hashing was largely a heuristic, and one used fixed functions that were empirically found to work well in cases where truly random functions could be shown to work well.

*Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

†Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.

The gap between hashing algorithms and their analysis narrowed with the advent of *universal hashing* [7]. 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. Following universal hashing, many hash function families have been proposed (e.g., [3, 6, 8, 10, 11, 12, 14, 16, 18, 21]), and their performance analyzed in various settings.

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 $\mathcal{H}$ of functions, $\mathcal{H}$ is called *k-wise independent.* The hash functions described by Carter and Wegman in [7], for example, were 2-wise independent. The first constructions of k-wise independent families required time $\Omega(k)$ for evaluating a hash function. (Here, and in the rest of the paper, we will consider complexity on a RAM with word size $\Theta(\log|U| + \log|V|)$.) A breakthrough was made by Siegel [21], who showed that high independence is achievable with relatively small families of hash functions that can be evaluated 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 lower bound on the number of bits needed to achieve k-wise independence is $\Omega(k)$ words [2, 9], and there are constructions using $O(k)$ words of space in the case where $|U|$ and $|V|$ are powers of the same prime. Sometimes there is a trade-off between the space used to represent a function and its evaluation time. For example, Siegel's k-wise independent family requires $k^{1+\Omega(1)}$ words of space to achieve constant evaluation time, for $|U| = k^{O(1)}$.

If one applies Siegel's 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 superlinear space usage means that, in many possible applications, the hash function description itself becomes asymptotically larger than all other parts of the data structure. 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 improves space to the optimal bound of $O(n)$ words.

Theorem 1 *Let $S \subseteq U$ be a set of n elements. For any constant $c > 0$ there is an algorithm constructing a random family of functions from U to V in $o(n)$ time and space, such that:*

- *With probability $1 - O(n^{-c})$ the family is uniform on S.*
- *There is a data structure of $O(n)$ words words representing its functions such that function values can be computed in constant time. The data structure can be initialized to a random function in $O(n)$ time.*

1.1 Implications

The fact that the space usage 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 [15] and Knuth [17]. 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[1]. While our family is not likely to 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., [19, 20]).

In addition to the classic hashing schemes, our hash functions provide a provably efficient implementation of a recent load balancing scheme of Azar et al. [4].

1.2 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 construction and its analysis. Finally, section 4 gives a number of applications of our result.

2 Background

The main result of this paper can be seen as an extension of Siegel’s family of high performance hash functions [21, 22]. 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 1 *A family $\mathcal{H}$ of functions from U to V is* k-wise independent *if, for any set of 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,*

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

[1]One could argue that hashing will always be a heuristic on real, deterministic machines. However, cryptographic applications have made it increasingly common to equip computers with a hardware random number generator, such as in Intel’s 8xx chipsets.

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. In this paper we will assume that the range V of hash functions is the set of elements in some group $R = (V, \oplus)$, where the group operation $\oplus$ can be performed in constant time on a RAM. There are many such examples of groups, for example those with group operations addition modulo $|V|$ and bitwise exclusive or.

Siegel primarily considered the case in which $|U| = k^{O(1)}$. He showed that in this case one can, for arbitrary constants $c, \epsilon > 0$, construct, in $o(k)$ time and space, a family of functions from U to V such that:

- The family is k-wise independent with probability $1 - k^{-c}$.
- There is a data structure of $k^{1+\epsilon}$ words words representing its functions such that function values can be computed in constant time. The data structure can be initialized to a random function in $k^{1+\epsilon}$ time.

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 [1].)

It is no coincidence that Siegel achieves constant evaluation time only for $|U| = k^{O(1)}$. He shows the following trade-off between evaluation time and the size of the data structure:

Theorem 2 (Siegel [21]) *For any k-wise independent family $\mathcal{H}$ of functions from U to V, any data structure using m words of $O(\log |V|)$ bits to represent a function from $\mathcal{H}$ requires worst case time $\Omega(\min(\log_{m/k}(|U|/k), k))$ to evaluate a function.*

Note that when using optimal space, i.e., $m = O(k)$, one must use time $\Omega(\min(\log(|U|/k), k))$ to evaluate a function. Siegel's upper bound extends to the case where $|U|$ is not bounded in terms of k. However, in this case the lack of an explicit expander construction results in an exponentially larger evaluation time than in the first term of the lower bound.

Theorem 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, say, 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' can be made low by choosing $|U'| = n^{O(1)}$ sufficiently large.

Definition 2 *A family of functions defined on U is* uniform *on the set $S \subseteq U$ if its restriction to S is $|S|$-wise independent.*

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

Theorem 3 (Siegel [21, 22]) *Let $S \subseteq U$ be a set of $n = k^{O(1)}$ elements. For any constants $\epsilon, c > 0$ there is an algorithm constructing a random family $\mathcal{SI}(U, V, k, n, c, \epsilon)$ of functions from U to V in $o(k)$ time and space, such that:*

- *With probability $1 - n^{-c}$ the family is k-wise independent on S.*
- *There is a data structure of $O(k^{1+\epsilon})$ words words representing its functions such that function values can be computed in constant time. The data structure can be initialized to a random function in $O(k^{1+\epsilon})$ time.*

With current expander "technology", Siegel's construction exhibits high constant factors. Other proposals for high performance hash functions, due to Dietzfelbinger and Meyer auf der Heide [12, 13], appear more practical. However, these families only exhibit $O(1)$-wise independence and appear to be difficult to analyze in general.

3 Hash function construction

In this section we describe our hash function family and show Theorem 1. We use the notation $T[i]$ to denote the ith entry in an array (or vector) T. By $[m]$ we denote the set $\{1, \ldots, m\}$.

3.1 The hash function family

Definition 3 *Let $R = (V, \oplus)$ be a group, let $\mathcal{G}$ be a family of functions from U to V, and let $f_1, f_2 : U \to [m]$. We define the family of functions*

$$\mathcal{H}(f_1, f_2, \mathcal{G}) = \{x \mapsto T_1[f_1(x)] \oplus T_2[f_2(x)] \oplus g(x) \mid T_1, T_2 \in V^m \text{ and } g \in \mathcal{G}\}.$$

A similar way of constructing a function family was presented in [12]. The novel feature of the above definition is the use of *two* values looked up in tables, rather than just one. The hash function family used to prove Theorem 1 uses Siegel's construction of function families to get the functions f_1 and f_2 and the family $\mathcal{G}$ in the above definition.

Definition 4 *For $n \leq |U|$ and any constant $c > 0$ we define the random family $\mathcal{H}_{n,c} = \mathcal{H}(f_1, f_2, \mathcal{G})$ of functions as follows: Construct the random families $\mathcal{G} = \mathcal{SI}(U, V, \sqrt{n}, n, c, 1/2)$ and $\mathcal{F} = \mathcal{SI}(U, [4n], \sqrt{n}, n, c, 1/2)$ according to Theorem 3, and pick f_1 and f_2 independently at random from $\mathcal{F}$.*

3.2 Properties of the family

For a set $S \subseteq U$ and two functions $f_1, f_2 : U \to [m]$ let $G(f_1, f_2, S) = (A, B, E)$ be the bipartite graph with vertex sets $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_m\}$, and edge set $E = \{e_x = (a_{f_1(x)}, b_{f_2(x)}) \mid x \in S\}$, where e_x is labeled by x. Note that there may be parallel edges.

We define a *cyclic 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 *cyclic part* $C \subseteq E$ is the maximal cyclic subgraph in the graph, i.e., the edges in cycles and edges in paths connecting cycles.

Lemma 1 *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 cyclic part of $G(f_1, f_2, S) = (A, B, E)$ is at most k, then $\mathcal{H}(f_1, f_2, \mathcal{G})$ is uniform on S.*

Proof. Let S' be the set of all elements $x \in S$ where the corresponding edge e_x is in the cyclic part C of $G(f_1, f_2, S)$.

The proof is by induction. First, assume that $|E \setminus C| = 0$. Since $S = S'$ and g is chosen from a k-wise independent family and $|S'| \leq k$ we can conclude that $\mathcal{H}(f_1, f_2, \mathcal{G})$ is uniform on S.

It remains to show that $\mathcal{H}(f_1, f_2, \mathcal{G})$ is uniform on S when $|E \setminus C| \geq 1$. Among the edges in $E \setminus C$ there has to be at least one edge with one unique endpoint. Let $e_{x^*} = (a_{f_1(x^*)}, b_{f_2(x^*)})$ be such an edge, $x^* \in S \setminus S'$. W.l.o.g. assume that $a_{f_1(x^*)}$ is the unique endpoint. By induction it holds that $\mathcal{H}(f_1, f_2, \mathcal{G})$ is uniform on $S \setminus \{x^*\}$. For $h \in \mathcal{H}(f_1, f_2, \mathcal{G})$ chosen at random, all values $h(x)$ for $x \in S \setminus \{x^*\}$ are independent of the value $T_1[f_1(x^*)]$. Additionally, given $g \in \mathcal{G}$ and all entries in vectors T_1 and T_2 except $T_1[f_1(x^*)]$, $h(x^*)$ is uniformly distributed when choosing $T_1[f_1(x^*)]$ at random. Hence $\mathcal{H}(f_1, f_2, \mathcal{G})$ is uniform on S.

□

Lemma 2 *For each set S of size n, and for $f_1, f_2 : U \to [4n]$ chosen at random from a family that is k-wise independent on S, $k \geq 32$, the probability that the cyclic part C of the graph $G(f_1, f_2, S)$ has size at least k is $n/2^{\Omega(k)}$.*

Proof. Assume that $|C| \geq k$ and that k is even (w.l.o.g.). Either there is a *connected* cyclic subgraph in $G(f_1, f_2, S)$ of size at least $k/2$ or there is a cyclic subgraph of size k', where $k/2 < k' \leq k$. In the first case there is a connected subgraph in $G(f_1, f_2, S)$ 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 $G(f_1, f_2, S)$, where $k/2 < k' \leq k$.

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 $G(f_1, f_2, S)$. Hence, we also get an upper bound on the probability that $|C|$ is at least k. Note that since f_1 and f_2 are chosen from a k-wise independent family, each subset of at most k edges will be random and independent. We will only consider subgraphs with at most k edges.

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':

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

□

Proof of Theorem 1. We will show that the random family $\mathcal{H}_{n,c}$ of Definition 4 fulfills the requirements in the theorem. Assume w.l.o.g. that $\sqrt{n}$ is integer. The families $\mathcal{G} = \mathcal{SI}(U, V, \sqrt{n}, n, c, 1/2)$ and $\mathcal{F} = \mathcal{SI}(U, [4n], \sqrt{n}, n, c, 1/2)$ are both $\sqrt{n}$-wise independent with probability $1 - n^{-c}$ for sets of size up to n according to Theorem 3. If $\mathcal{F}$ is $\sqrt{n}$-wise independent then by Lemma 2 the probability that the cyclic part of graph $G(f_1, f_2, S)$ has size at most $\sqrt{n}$ is at least $1 - n^{-\Omega(\sqrt{n})}$, if $\sqrt{n} \geq 32$. We can assume w.l.o.g. that $\sqrt{n} \geq 32$, since otherwise the theorem follows directly from Theorem 3. When the cyclic part of graph $G(f_1, f_2, S)$ has size at most $\sqrt{n}$ then, by Lemma 1, $\mathcal{H}_{n,c}$ is uniform on S if $\mathcal{G}$ is $\sqrt{n}$-wise independent. The probability that $\mathcal{G}$ is $\sqrt{n}$-wise independent, $\mathcal{F}$ is $\sqrt{n}$-wise independent, and that the cyclic part of graph $G(f_1, f_2, S)$ has size at most $\sqrt{n}$ is altogether $(1 - n^{-c})^2(1 - n^{-\Omega(\sqrt{n})}) = 1 - O(n^{-c})$.

The construction of $\mathcal{H}_{n.c}$, i.e., constructing $\mathcal{F}$ and $\mathcal{G}$ and choosing f_1 and f_2, can according to Theorem 3, be done in time and space $o(n)$. The space usage of a data structure representing a function from $\mathcal{H}_{n,c}$ is $O(n)$ words for T_1 and T_2, and $o(n)$ words for storing $g \in \mathcal{G}$. The initialization time is dominated by the time used for initializing T_1 and T_2 to random arrays. Function values can clearly be computed in constant time. $\square$

4 Applications

We now characterize a class of data structures that, when used with our hash function construction, behave exactly as if uniform hashing was used, in the sense that at any time it holds (with high probability) that the probability distribution over possible memory configurations is the same. We give a number of examples of data structures falling into this class.

Definition 5 *A data structure with oracle access to a hash function $h : U \to V$ is n-*hash-dependent *if there is a function f mapping operation sequences to subsets of U of size at most n, such that after any sequence of operations $O_1, \ldots, O_t$, the memory configuration depends only on $O_1, \ldots, O_t$, the random choices made by the data structure, and the function values of h on the set $f(O_1, \ldots, O_t)$.*

The following is an immediate implication of Theorem 1.

Theorem 4 *Consider a sequence of $n^{O(1)}$ operations on an n-hash-dependent RAM data structure with a random hash function oracle. For any constant $c > 0$, the oracle can be replaced by a random data structure using $O(n)$ words of space and increasing time by at most a constant factor, such that with probability $1 - O(n^{-c})$ the distribution of memory configurations after each operation remains the same.*

At first glance, the theorem concerns only what the data structure will look like, and does not say anything about the behavior of queries. However, in most cases $O(n)$-hash-dependence is maintained if we extend a data structure to write down in memory, say, the memory locations inspected during a query. Using the theorem on this data structure one then obtains that also the distribution of memory accesses during queries is preserved when using our class of hash functions.

The additional space usage of $O(n)$ words can be reduced if U is much larger than V by packing several $\log |V|$ bit entries of the arrays T_1 and T_2 in each $\Theta(\log |U|)$ bit word. It should be noted that although $O(n)$ words may be of the same order as the space used by the rest of the data structure, there are many cases where it is negligible. For example, if more than a constant number of words of associated information is stored with each key in a hash table, the space usage for our hash function is a vanishing part of the total space.

4.1 Examples

In the following we describe some n-hash-dependent hashing schemes.

Insertion only hash tables. One class of hash tables that are clearly n-hash-dependent are those that support only insertions of elements, have a bound of n on the number of elements that can be inserted (before a rehash), and use h only on inserted elements. This is the primary kind of scheme considered by Gonnet in [15], and includes linear probing, double hashing, quadratic hashing, ordered hashing, Brent's algorithm, chained hashing, coalesced hashing, and extendible hashing.

Many such schemes are extended to support deletions by employing "deletion markers". However, as noted by Knuth [17], deleting many elements in this way tends to lead to very high cost for unsuccessful searches. It thus makes sense to rebuild such data structures (with a new hash function) when the *total* number of insertions and deletions reaches some number n (around the size of the hash table). If this is done, the hashing scheme remains n-hash-dependent.

Deletion independent hash tables. Some hash tables have the property that deleting an element x leaves the data structure in exactly the state it would have been in if x had never been inserted. In particular, the state depends exclusively on the current set of elements, the order in which they were inserted, and their hash function values. If the capacity of the hash table is bounded by n, such a data structure is n-hash-dependent.

An example of the above is a hash table using linear probing, with the deletion algorithm in [17]. Also, chained hashing methods have deletion independent *pointer structure*. In particular, for those methods we get n-hash-dependence up to pointer structure equivalence.

Load balancing. A load balancing scheme of Azar et al. [4], further developed and analyzed in [5, 23], can also be thought of as a hashing data structure. This scheme has been analyzed under the uniform hashing assumption. It has the property that an element in the hash table never needs to be moved once it has been placed, while at the same time, the worst case time for accessing an element remains very low.

Theorem 4 implies that, in the insertion only case, this data structure can be efficiently implemented such that the uniform hashing analysis holds with high probability. This means, in turn, that this is also true for the load balancing scheme.

References

[1] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

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

[3] Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos. Is linear hashing good? In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97)*, pages 465–474. ACM Press, 1997.

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

[5] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: the heavily loaded case. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, pages 745–754. ACM Press, 2000.

[6] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *J. Comput. System Sci.*, 60(3):630–659, 2000.

[7] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. System Sci.*, 18(2):143–154, 1979.

[8] Andrew Chin. Locality-preserving hash functions for general purpose parallel computation. *Algorithmica*, 12(2-3):170–181, 1994.

[9] 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)*, pages 396–407. IEEE Comput. Soc. Press, 1985.

[10] 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)*, pages 569–580. Springer-Verlag, 1996.

[11] Martin Dietzfelbinger, Joseph Gil, Yossi Matias, and Nicholas Pippenger. Polynomial hash functions are reliable (extended abstract). In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, volume 623 of *Lecture Notes in Computer Science*, pages 235–246. Springer-Verlag, 1992.

[12] 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)*, volume 443 of *Lecture Notes in Computer Science*, pages 6–19. Springer-Verlag, 1990.

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

[14] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997.

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

[16] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97)*, pages 618–625. ACM, New York, 1999.

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

[18] Nathan Linial and Ori Sasson. Non-expansive hashing. *Combinatorica*, 18(1):121–132, 1998.

[19] 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)*, pages 355–366. ACM Press, 1989.

[20] Jeanette P. Schmidt and Alan 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)*, pages 224–234. ACM Press, 1990.

[21] Alan Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS '89)*, pages 20–25. IEEE Comput. Soc. Press, 1989.

[22] Alan Siegel. On universal classes of extremely random constant time hash functions and their time-space tradeoff. Technical Report TR1995-684, New York University, April, 1995.

[23] Berthold Vöcking. How asymmetry helps load balancing. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*, pages 131–141. IEEE Computer Society Press, 1999.

Recent BRICS Report Series Publications