Design Strategies for Minimal Perfect Hash Functions

Martin Dietzfelbinger

Technische Universität Ilmenau, 98684 Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Abstract. A minimal perfect hash function h for a set $S \subseteq U$ of size n is a function $h: U \to \{0, \ldots, n-1\}$ that is one-to-one on S. The complexity measures of interest are storage space for h, evaluation time (which should be constant), and construction time. The talk gives an overview of several recent randomized constructions of minimal perfect hash functions, leading to space-efficient solutions that are fast in practice. A central issue is a method ("split-and-share") that makes it possible to assume that fully random (hash) functions are available.

1 Introduction

In this survey paper we discuss algorithmic techniques that are useful for the construction of minimal perfect hash functions. We focus on techniques for managing randomness.

We assume a set $U = \{0,1\}^w$ (the "universe") of "keys" x is given. Assume that $S \subseteq U$ is a (given) set with cardinality $n = |S|$, and that $m \geq n$. A function $h: U \to [m]$ that is one-to-one on S is called a *perfect hash function* (for S). If in addition $n = m$ (the smallest possible value), h is called a *minimal perfect hash function* (MPHF).[1]

The MPHF problem for a given $S \subseteq U$ is to construct a data structure D_h that allows us to evaluate $h(x)$ for given $x \in U$, where h is a MPHF for S. The parameters of interest are the storage space for D_h and the evaluation time of h, which should be constant. Clearly, such a data structure D_h can be used to devise a (static) dictionary that for each key $x \in S$ stores x and some data item d_x in an array of size n, with constant retrieval time.

In the past decades, the MPHF problem has been studied thoroughly. For a detailed survey of the developments up to 1997 see the comprehensive study [9]. To put the results into perspective, one should notice the fundamental space lower bound of $n \log e + \log w - O(\log n)$ bits[2], valid as soon as $w \geq (2 + \varepsilon) \log n$, proved by Fredman and Komlós [18]. This bound is essentially tight: Mehlhorn [23, Sect. III.2.3, Thm. 8] gave a construction of a MPHF that takes $n \log e + \log w + O(\log n)$ bits of space (but has a vast evaluation time). In order not to have to worry about the influence of the size 2^w of U too much, unless

[1] $[m]$ denotes the set $\{0, \ldots, m-1\}$.
[2] All logarithms in this paper are to the base 2. Note that $\log e \approx 1.443\ldots$

© Springer-Verlag Berlin Heidelberg 2007

noted otherwise, we will assume in the following that $n > w \geq (2+\varepsilon)\log n$, and subsume the term $\log w$ in the space bounds in terms $O(\log n)$ and larger.

1.1 Space-Optimal, Time-Efficient Constructions

The (information-)theoretical background settled, the question is how close to the bound $n \log e + \log w$ one can get if one insists on constant evaluation time. In the seminal paper [19] Fredman, Komlós, and Szemerédi constructed a dictionary with constant lookup time, which can be used to obtain a MPHF data structure with constant evaluation time and space $O(n \log n)$ bits. Based on [19], Schmidt and Siegel [28] gave a construction for MPHF with constant evaluation time and space $O(n)$ bits (optimal up to a constant factor). Finally, Hagerup and Tholey [20] described a method that in expected linear time constructs a data structure D_h with $n + \log w + o(n + \log w)$ bits, for evaluating a MPHF h in constant time. This is space-optimal up to an additive term. It seems hard, though, to turn the last two constructions into data structures that are space efficient and practically time efficient at the same time for realistic values of n.

1.2 Practical Solutions

In a different line of development, methods for constructing MPHF were studied that emphasized the evaluation time and simple construction methods over optimality of space. Two different lines (a "graph/hypergraph-based approach" and a method called "hash-and-displace") in principle led to constructions of very simple structures that offered constant evaluation time and a space requirement that was dominated by a table of $\Theta(n)$ elements of $[n] = \{0, \ldots, n-1\}$, which means $\Theta(n \log n)$ bits. Very recently, refinements of these methods were proposed that lead to a space requirement of $O(n \log\log n)$ bits (and constant evaluation time) [11,32]. Only in 2007, Botelho, Pagh, and Ziviani [5] managed to devise a construction for a MPHF that is simple and time-efficient, and gets by with $O(n)$ bits of storage space, with a constant factor that is only a small factor away from the information theory minimum $\log e \approx 1.44$. Crucial steps in this development will be described in some detail in the rest of this paper.

1.3 Randomness Assumptions

Given a universe U of keys, a *hash function* is just any function $h: U \to [m]$. Most constructions of MPHF involve several hash functions, which must behave randomly in some way or the other. There are two essentially different ways to approach the issue of the hash functions:

The "full randomness" assumption: One assumes that a sequence $h_0, h_1, \ldots$ of hash functions is available, so that evaluating $h_i(x)$ takes constant time, no storage space is needed for these functions, and such that $h_i(x)$, $x \in S$, $i \geq 0$, are fully random values (uniform in $[m]$, independent). The analysis of several MPHF algorithms is based on this assumption (e. g., [8,22,7,4]).
Randomization: "Universal hashing" was introduced by Carter and Wegman [6] in 1979. One uses a whole set ("class") $\mathcal{H}$ of hash functions and chooses one such

function from $\mathcal{H}$ at random whenever necessary. Normally, some parameters of a function with a fixed structure are chosen at random. Storing the function means storing the parameters; the analysis is carried out on the basis of the probability space induced by the random choice of the function. Some classical MPFH algorithm use this approach (e. g., [28,25,20]).

Below, we will explain in detail how in the context of the MPHF problem one may quite easily work around the randomness issue by using very simple universal hash classes. To be concrete, we describe two such classes here. We identify $U = \{0,1\}^w$ with $[2^w]$.

Definition 1. *A set $\mathcal{H}$ of functions from U to $[m]$ is called* 1-universal *if for each pair of different $x, y \in U$ and for h chosen at random from $\mathcal{H}$ we have*

$$\Pr(h(x) = h(y)) \le \frac{1}{m}.$$

There are many constructions of 1-universal classes. One is particularly simple (see [6]): Assume p is a prime number larger than 2^w, and $m \le 2^w$. For $a, b \in [p]$ define $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$, and let $\mathcal{H}_m = \{h_{a,b} \mid a \in [p] - \{0\}, b \in [p]\}$. Choosing/storing a hash function from $\mathcal{H}_m$ amounts to choosing/storing the coefficients a and b (not much more than $2w$ bits).

Definition 2. *Let $k \ge 2$. A set $\mathcal{H}$ of functions from U to $[m]$ is called k-wise independent if for each sequence $(x_1, \ldots, x_k)$ of different elements of U and for h chosen at random from $\mathcal{H}$ we have that the values $h(x_1), \ldots, h(x_k)$ are fully random in $[m]^k$ and each value $h(x)$ is [approximately] uniformly distributed in $[m]$.*

The simplest way of obtaining a k-wise independent class is by using polynomials. Let $p > 2^w$ be a prime number as before, and let $m^{1+\varepsilon} \le 2^w$ for some $\varepsilon > 0$. The set $\mathcal{H}_m^k$ of all functions of the form

$$h(x) = ((a_{k-1}x^{k-1} + \cdots + a_1 x + a_0) \bmod p) \bmod m, \quad a_{k-1}, \ldots, a_0 \in [p]$$

(polynomials over the field $\mathbf{Z}_p$ of degree smaller than k, projected into $[m]$), is k-wise independent. Choosing/storing a hash function amounts from this class amounts to choosing/storing the coefficients $(a_{k-1}, \ldots, a_0)$. For details see, e. g., [15,12]. The evaluation time is $\Theta(k)$. For more sophisticated hash function constructions see e. g. [29,14,30].

2 Split-and-Share for MPHFs

Let $S \subseteq U$ be fixed, $n = |S|$. For a hash function $h\colon S \to [m]$ and $i \in [m]$ let $S_i = \{x \in S \mid h(x) = i\}$, and let $n_i = |S_i|$. It is a common idea, used many times before in the context of perfect hashing constructions (e. g. in [19,20,10]), to construct separate and disjoint data structures for the "chunks" S_i.

The new twist is to "share randomness" among the chunks S_i, as follows. (The approach was sketched, for different applications, in [17,16].) In the static setting, with S given, this works as follows: Choose h, and calculate the sets $S_i = \{x \in S \mid h(x) = i\}$ and their sizes n_i, repeating if necessary until the sizes are suitable. Then devise one data structure that for each i provides one or several hash functions that behave fully randomly on S_i. Each S_i may own some component of this data structure but one essential part (usually a big table of random words) is used ("shared") by all S_i's.

We describe the approach in more detail. First, we "split", and make sure that none of the chunks is too large. The proof of the following lemma is standard.

Lemma 1. *If $m \geq 2n^{2/3}$ and $h\colon U \to [m]$ is chosen at random from a 4-universal class $\mathcal{H} = \mathcal{H}_m^4$, then* $\Pr(\max\{|S_i| \mid 0 \leq i < m\} > \sqrt{n}) \leq \frac{1}{4}$.

Proof. The probability that $|S_i| > \sqrt{n}$ is bounded by

$$\Pr\Big(\binom{|S_i|}{4} \geq \binom{\sqrt{n}}{4}\Big) \leq \frac{\mathrm{E}(\binom{|S_i|}{4})}{\binom{\sqrt{n}}{4}} \leq \frac{\binom{n}{4}/(2n^{2/3})^4}{\binom{\sqrt{n}}{4}} < \frac{1}{8n^{2/3}},$$

for n large enough; hence $\Pr(\exists i\colon |S_i| \geq \sqrt{n}) \leq 2n^{2/3}/(8n^{2/3}) = \frac{1}{4}$.

Given S, we fix $m = 2n^{2/3}$ and repeatedly choose h from $\mathcal{H}_m^4$ until an h with $\max\{|S_i| \mid 0 \leq i < m\} \leq \sqrt{n}$ is found. We fix this function h and call it h^0 from here on; thus also the S_i and the n_i are fixed. With $a_i = \sum_{0 \leq j < i} n_i$ we can allocate indices in the interval $[a_i, a_{i+1} - 1]$ as possible hash values for keys in S_i.

Once we have found MPHFs h_i, one for each S_i, we may let

$$h(x) = a_i + h_i(x) \text{ for } i = h^0(x), \tag{1}$$

thus obtaining an MPHF for all of S. Below, we will describe several methods for building such a MPHF h_i. For this, it is most convenient to have at our disposal one or several hash functions that behave fully randomly (on each S_i separately). To make this concrete, let $K > 1$ be some constant, and let $L = K \log n$. We will argue that when considering S_i we may assume that we have a source of L fully random hash functions $h_1, \ldots, h_L$ from U to $\{0,1\}^k$ for some k we may choose, which can be evalutated in (small) constant time. The data structure that provides the random elements used in these functions will be shared among the different h_i.

Let $\mathcal{H}_r$ denote an arbitrary 1-universal class of functions from U to $[r]$.

Lemma 2. *Let $r = 2n^{3/4}$. For an arbitrary given $S' \subseteq U$ with $n' = |S'| \leq \sqrt{n}$ we may in expected time $O(|S'|)$ find two hash functions h_0, h_1 from $\mathcal{H}_r$ such that for any two tables $T_0[0..r-1]$ and $T_1[0..r-1]$, each containing r random elements from $\{0,1\}^k$, we have that $h'(x) = T_0[h_0(x)] \oplus T_1[h_1(x)]$ defines a function $h' : U \to \{0,1\}^k$ that is fully random on S'.* ($\oplus$ denotes bitwise XOR.)

Proof. Assume h_0, h_1 are chosen at random from $\mathcal{H}_r$. We call a pair h_0, h_1 *good* if for each $x \in S'$ there is some $i \in \{0,1\}$ such that $h_i(x) \neq h_i(y)$ for all $y \in S' - \{x\}$. For each $x \in S'$, the probability that $\exists y_0 \in S' - \{x\}\colon h_0(x) = h_0(y_0)$

and $\exists y_1 \in S' - \{x\}\colon h_1(x) = h_1(y_1)$ is smaller than $(\sqrt{n}/r)^2 \leq 1/(4\sqrt{n})$. This implies that the probability that (h_0, h_1) is not good is bounded by $\frac{1}{4}$. We keep choosing h_1, h_2 from $\mathcal{H}_r$ until a good pair is found — the expected number of trials is smaller than $\frac{4}{3}$. Checking one pair h_1, h_2 takes time $O(|S'|)$ when utilizing an auxiliary array of size r. Once a good pair h_1, h_2 has been fixed, for a key $x \in S'$ either table position $T_0[h_0(x)]$ or table position $T_1[h_1(x)]$ appears in the calculation of $h(x)$ but of no other key $y \in S'$. Since this entry is fully random, and because $\{0,1\}^k$ with $\oplus$ is a group, $h(x)$ is random and independent of the other hash values $h(y)$, $y \in S' - \{x\}$.

From here, we proceed as follows: For each i, $0 \leq i < m$, we choose hash functions h_0^i, h_1^i that are as required in Lemma 2 for $S' = S_i$. The descriptions of these $2m$ hash functions as well as the sizes n_i and the offsets a_i can be stored in (an array that takes) space $O(m) = O(n^{3/4})$ (words of length $O(w)$).

Now we describe the "shared" part of the data structure: Recall that $L = K \log n$. For each $j \in [L]$ we initialize arrays $T_{j,0}[0..r-1]$ and $T_{j,1}[0..r-1]$ with random words from $\{0,1\}^k$. We let

$$h_j^i(x) = T_{j,0}[h_0^i(x)] \oplus T_{j,1}[h_1^i(x)], \text{ for } x \in U,\ 0 \leq j < L,\ 0 \leq i < m.$$

Since h_0^i, h_1^i satisfy the condition in Lemma 2, for each fixed i we have that the values $h_{i,j}(x), x \in S_i, j \in [L]$, are fully random. The overall data structure takes up space $2n^{3/4} \cdot L$ words from $\{0,1\}^k$ plus $O(n^{2/3})$ words of size $\log |U|$, for the description of the h_0^i, h_1^i. We will see below that with high probability these hash functions will be sufficient for constructing a MPHF h_i for S_i, for all $i \in [m]$. If that construction is not successful, we start all over, with new random entries in the arrays $T_{j,0}$ and $T_{j,1}$.

From here on we *assume* that we have a fixed set S' of size $n' \leq \sqrt{n}$ and a supply of $L = K \log n$ fully random hash functions $h_0, \ldots, h_{L-1}$ with constant evaluation time and range $\{0,1\}^k$ (identified with $[2^k]$).

Goal: Build a MPHF for S' that has constant evaluation time and requires little storage space (beyond the functions $h_0, \ldots, h_{L-1}$). In the rest of the paper we discuss various strategies for achieving this.

3 Hash-and-Displace Approach

In this section, we discuss an approach to obtaining a MPHF by splitting S' into buckets, hashing the buckets into the common range $[n']$ and adjusting by offsets.

3.1 Pure Hash-and-Displace

Pagh [25] introduced the following approach for constructing a minimal perfect hash function for a set S': Choose hash functions $f\colon U \to [n']$ and $g\colon U \to [m']$. The set $[m'] \times [n']$ may be thought of as an array A with entry at (i, j) equal to 1 if $(f(x), g(x)) = 1$ for some $x \in S$, and 0 otherwise. Let $B_i = \{x \in S' \mid g(x) = i\}$,

$0 \le i < m'$. We would like to see that f when restricted on B_i distributes the keys one-to-one into (the ith copy of) $[n']$. Technically, we check whether (f, g) is one-to-one on S' and whether

$$\sum_{\substack{0 \le i < m' \\ |B_i| \ge 2}} |B_i|^2 < (1-\delta)n'. \tag{2}$$

for some constant $\delta > 0$. Inequality (2) implies the "harmonic decay property" which is at the heart of the analysis of Pagh's algorithm:

$$s \cdot \sum_{\substack{0 \le i < m' \\ |B_i| \ge s}} |B_i| < (1-\delta)n', \text{ for all } s \ge 2. \tag{3}$$

If (2) is not satisfied, choose new (f, g) until (2) is satisfied. Pagh showed that once (2) is guaranteed a simple randomized scheme RFD ("random fit decreasing") in expected time $O(n')$ finds "displacements" $d_i \in [n']$, $0 \le i < m'$, such that the function

$$h(x) = (f(x) + d_{g(x)}) \bmod n' \tag{4}$$

is (minimal) perfect for S'. Here, RFD works as follows: Sort the "rows" B_i by falling "weight" $|B_i|$. In this order, treat rows with $|B_i| \ge 2$ as follows: Repeat choosing d_i at random from $[n']$ until $\{(f(x) + d_i) \bmod n' \mid x \in B_i\}$ is disjoint from $\{(f(x) + d_{g(x)}) \bmod n' \mid x \in B_{i'}, B_{i'} \text{ already placed}\}$. Rows B_i with $|B_i| = 1$ are placed in one final deterministic round.

The question is how small m' may be chosen so that functions f and g as required can be found. Pagh based his construction on simple 1-universal classes and showed that to get by with f and g from such classes it is sufficient to have $m' > (2+\varepsilon)n'$. Looking at (3) it is easy to see [13] that in place of (2) the following conditions are sufficient to make sure that RFD works: (f, g) is one-to-one on S' and

$$2 \cdot \sum_{\substack{0 \le i < m' \\ |B_i| \ge 2}} |B_i| < (1-\delta)n' \quad \text{and} \quad \sum_{\substack{0 \le i < m' \\ |B_i| \ge 3}} |B_i|^2 < (1-\delta)n'. \tag{5}$$

for some constant $\delta > 0$. With $\alpha = n'/m'$ one may show (with techniques explained in more detail in [13]) that asymptotically $(n', m' \to \infty)$

$$\mathrm{E}\Big(\sum_{\substack{0 \le i < m' \\ |B_i| \ge 2}} |B_i|\Big) \approx n' \cdot (1 - e^{-\alpha}) \quad \text{and} \quad \mathrm{E}\Big(\sum_{\substack{0 \le i < m' \\ |B_i| \ge 3}} |B_i|^2\Big) \approx n' \cdot (\alpha + 1 - e^{-\alpha} - 2\alpha e^{-\alpha}). \tag{6}$$

Since for suitable $\varepsilon > 0$ and $\alpha < \ln 2/(1+\varepsilon)$ we have $2(1 - e^{-\alpha}) < 1 - \delta$ and $\alpha + 1 - e^{-\alpha} - 2\alpha e^{-\alpha} < 0.6 < 1 - \delta$, this means that for n', m' large enough and $m' > 1.45(1+\varepsilon)n' > (1+\varepsilon)(\log e)n'$, inequalities (5) will be satisfied with high probability. Since the probability that (f, g) is not one-to-one on S' can be bounded by $\binom{n'}{2}/(n'm') < 1/(2 \cdot 1.45)$, a random pair (f, g) will be suitable with probability larger than 1/3. If we try $(f, g) = (h_{2t} \bmod n', h_{2t+1} \bmod m'), t = 0, 1, \ldots, L/2-1$, for being one-to-one on S' and (5) being satisfied, the expected number of trials will be not larger than 3, the expected time will be $O(|S'|)$, and the probability we are not finished after testing $L/2$ pairs is at most $3^{-L/2} = 3^{-(K/2)\log n} = n^{-(K\log 3)/2}$. We can make this smaller than n^{-3} by choosing K large enough. Thus the probability that for some i no suitable (f, g) is found among $(h_{2t} \bmod n_i, h_{2t+1} \bmod m_i)$, $t = 0, 1, \ldots, L/2-1$, is bounded by $m \cdot n^{-3} \leq n^{-2}$. (In this improbable case we choose new random entries for $T_{j,0}$ and $T_{j,1}$ and start all over.)

The overall data structure D_h for a hash-and-displace MPFH h for the whole set S consists of the following pieces:

- the splitting hash function h^0;
- h^i_0, h^i_1, for $0 \leq i < m$;
- values a_i (and b_i), for $0 \leq i < m$;
- arrays $T_{j,0}[0..r-1]$ and $T_{j,1}[0..r-1]$, for $j \in [L]$;
- an index $t \in [L/2]$ for the suitable pair $(h_{2t} \bmod n_i, h_{2t+1} \bmod m_i)$;
- $1.45(1+\varepsilon)n_i$ offset values in $[n_i]$, for $0 \leq i < m$.

The overall space needed is $2m = 4n^{2/3}$ words of size $\log|U|$ and $mL = O(n^{2/3} \cdot \log n)$ words from $\{0,1\}^k$ and $1.45(1+\varepsilon)n$ offset values in $[\sqrt{n}]$ (about $0.78n \log n$ bits).

As remarked by P. Sanders [27], the space requirements may be lowered further asymptotically by increasing m to some larger power $n^{(t-1)/t}$, and increasing the degree of the splitting hash function h^0.

Remark: For the RFD algorithm to work, it is necessary that (f, g) are one-to-one, but condition (5) is only sufficient, not necessary. It is interesting to note that in (preliminary) experiments values of m' down to below $0.3n'$ still seem to work, so in a supervised situation where a MPHF is to be built (and one might resort to the pure, certified algorithm if not successful) it may save up to two thirds of the space if one tries to run RFD without checking (5).

3.2 Undo-One

Dietzfelbinger and Hagerup [13] modified Pagh's approach [25] as follows: Functions (f, g) were chosen to be one-to-one and satisfy the following conditions:

$$\sum_{\substack{0 \leq i < m' \\ |B_i| \geq 3}} |B_i|^2 < (1-\delta)n' \quad \text{and} \quad |\{i \mid B_i \neq \emptyset\}| + |\{i \mid |B_i| = 1\}| \geq (1+\delta)n'. \quad (7)$$

Once condition (7) is satisfied, a variant of Pagh's algorithm can be proved to find suitable offsets in expected constant time: For sets B_i with $|B_i| \geq 3$ run the RFD algorithm as before; for sets B_i of size 2 it is also checked whether they can be successfully be placed by moving up to one set that was placed before (for details see [13]).

It can be shown (using techniques from [13]) that for (f, g) fully random functions and $m' = (1+\varepsilon)n'$, for an arbitrary fixed $\varepsilon > 0$, relation (7) holds with high probability, as long as n' is sufficiently large. This means that we may search for (f, g) just as in the previous section — only checking for (7) to hold. The final data structure will look the same as in the previous section. In contrast to [13], where for smaller ε polynomials of larger and larger degree are employed, the evaluation time for the hash functions described here does not depend on ε anymore. The overall space needed is $2m = 4n^{2/3}$ words of size $\log |U|$ and $n^{2/3}L = O(n^{2/3} \log n)$ words from $\{0, 1\}^k$ and $(1 + \varepsilon)n$ offset values in $[\sqrt{n}]$ (a little more than $0.5n \log n$ bits). Again, the constant factor in front of the $n \log n$ may be reduced at the expense of increasing m and the degree of independence of h^0.

4 Minimal Perfect Hashing by the Multifunction Paradigm

A different approach to constructing MPHF uses several hash functions. In that, it resembles the approach taken in the area of Bloom filters (see [3], and e. g. [7]).

4.1 The Hypergraph Approach

Czech *et al.* [8] and Majewski *et al.* [22] introduced the following approach to constructing a MPHF. To each key x associate a sequence $(h_1(x), \ldots, h_d(x))$ of *distinct* hash values in some range $[m]$. The structure consisting of $V = [m]$ and the system of (labeled) sets $e_x = \{h_1(x), \ldots, h_d(x)\}$, $x \in S'$, may be regarded as a hypergraph $G(S', h_1, \ldots, h_d)$ of order (edge size) d. If the elements of S' can be arranged in a sequence $(x_1, \ldots, x_{n'})$ such that

$$e_{x_j} - \bigcup_{s<j} e_{x_s} \neq \emptyset \text{ , for } j = 1, \ldots, n' \qquad (8)$$

then we say that $G(S', h_1, \ldots, h_d)$ is *acyclic*. It is useful to consider the vertex-edge incidence matrix A_G of $G(S', h_1, \ldots, h_d)$. It has n' rows, labeled with $x_1, \ldots, x_{n'}$, where position ℓ in row j is 1 if $\ell \in \{h_1(x_j), \ldots, h_d(x_j)\}$, and is 0 otherwise. Condition (8) entails that in this matrix in row j there is a 1-entry in some position ℓ_j so that column ℓ_j has only 0s above row j. Thus the matrix A_G can be transformed into echelon form by exchanging columns. This immediately implies the following.

Lemma 3. *If* (8) *holds, then for each vector* $(b_1, \ldots, b_{n'}) \in [n']$ *we may* (*even in linear time*) *find a set of values* $g(i) \in [n']$, $i \in [m']$, *such that*

$$(g(h_1(x_j)) + \cdots + g(h_d(x_j))) \bmod m' = b_j \text{ , for } 1 \leq j \leq n'.$$

It can be arranged that $g(i) = 0$ *for* $i \notin \{\ell_j \mid 1 \leq j \leq n'\}$. *If we choose* $(b_1, \ldots, b_{n'})$ *as a permutation of* $(0, 1, \ldots, n' - 1)$, *we obtain a MPHF for* S'.

Remark 1. In [7] the approach of the acyclic hypergraph was re-discovered and utilized in a similar way as in [22] to implement an arbitrary function $f\colon S' \to \{0, 1\}^q$, even including a mechanism to detect (with some probability $1 - \varepsilon$)

if a key $x \notin S'$ is presented to the data structure. (This problem was called "retrieval" in [10].) In [7], a naive analysis of the acyclicity property was used, leading to an estimate $O(n')$ of the space requirements that is much larger than the space bounds from [22], as discussed below. However, the bounds from [22] do apply also in this context.

In [8] and [22] it was assumed that fully random hash functions are available. This gap in the analysis vanishes if one employs the "split-and-share" trick.

A minor question that remains is how one may find a mapping $x \mapsto (h_1(x), \ldots, h_d(x))$ that attains vectors of d different elements as values, each one with the same probability, if only given fully random values $(h_1^0(x), \ldots, h_d^0(x))$ in $\{0,1\}^k$. There are several approaches to this problem, a solution due to Floyd being discussed in [2]. (A workaround used in some papers (e. g. [5]) is to let $h_1, \ldots, h_d$ have disjoint ranges of size m'/d each, the slight disandvantage being that results from the random graphs literature do not apply directly to this situation of "d-partite hypergraphs".)

Again, with the "splitting" approach at the basis, we do not have to worry about space, and can even simplify Floyd's method, utilizing an idea usually employed to construct (full) random permutations (see [21, Algorithm P]). Use an auxiliary array $R[0..\sqrt{n}-1]$, initialized so that $R[i] = i$ for all i (this property is restored after each use). We assume that a fully random sequence $h_1^0, \ldots, h_d^0$ of hash functions with range $[2^k]$ is available, $k \geq 2\log n$.

```
**Algorithm.** *Hyperedge*
**Input:** $x$, $n'$.
**Output:** $(h_1(x), \ldots, h_d(x))$ (* distinct values *)
**for** $\ell = 1$ **to** $d$ do
    $j_\ell \leftarrow h_\ell^0(x) \bmod (n' - \ell + 1)$;
    exchange $R[j_\ell]$ and $R[n' - \ell]$;
$(z_1, \ldots, z_d) \leftarrow (R[n'-1], \ldots, R[n'-d])$;
**for** $\ell = 1$ **to** $d$ do
    $R[j_\ell] \leftarrow j_\ell$; $R[n'-\ell] \leftarrow n' - \ell$;
**return** $(z_1, \ldots, z_d)$.
```

It is not hard to check that each d-tuple $(z_1, \ldots, z_d)$ in $[n']$ that consists of d distinct values (up to negligibly small rounding errors) has the same probability to be returned as $(h_1(x), \ldots, h_d(x))$. Once the edges e_x, $x \in S'$, have been calculated, one may easily in linear time calculate an ordering $(x_1, \ldots, x_{n'})$ that satisfies (8), if such an ordering exists. (For details see [22].) If no such ordering exists (the hypergraph $([m], \{e_x\}_{x \in S'})$ is "cyclic"), we repeat with a new set $h_1^0, \ldots, h_d^0$ of fully random hash functions. (If this approach is implemented in the context of the "split-and-share" approach, for each trial a new segment of d of the fully random functions $h_0, \ldots, h_{L-1}$ are used.)

In [22] it is discussed in detail what the probability for acyclicity is for various d and quotients $c = n'/m'$ (assuming the asymptotic case with $n', m' \to \infty$). For $d = 2$ we must have $c > 2$ and get an acyclicity probability of $e^{1/c}\sqrt{(c-2)/c} > 0$.

For $d = 3, 4, 5$ one gets threshold values

$$c_3 \approx 1.222, c_4 \approx 1.295, c_5 \approx 1.425,$$

meaning that if $n'/m' \geq c > c_d$ then the probability that the hypergraph is acylic is high (approaching 1 as n', m' grow). Larger values of d have worse threshold values. The most attractive choice for d obviously is $d = 3$, where a choice of $m' = 1.23n'$ leads to a good chance for hitting an acyclic hypergraph.

Thus, a data structure for a mapping $U \ni x \mapsto e_x = \{h_1(x), \ldots, h_d(x)\}$ so that $([m'], \{e_x\}_{x \in S'})$ forms an acyclic hypergraph can be constructed in expected time $O(|n'|)$. We have already seen how such a structure can be used to get a MPHF for S' that in essence consists of a table of m' numbers from $[\sqrt{n}]$. If one uses this construction for each set S_i separately, sharing the random entries in the arrays $T_{j,0}, T_{j,1}$ as before, one obtains a data structure D_h for a MPFH h for S that has the following components:

- the splitting hash function h^0;
- h_0^i, h_1^i, for $0 \leq i < m$;
- values a_i (and b_i), $0 \leq i < m$;
- arrays $T_{j,0}[0..r-1]$ and $T_{j,1}[0..r-1]$, $j \in [L]$;
- an index $t \in [L/2]$ for the suitable triple $(h_{3t} \bmod n_i, h_{3t+1} \bmod m_i, h_{3t+2} \bmod m_i)$;
- $1.23n_i$ g_i-values in $[n_i]$, for $0 \leq i < m$.

The overall space needed is $2m = 4n^{2/3}$ words of size $\log |U|$ and mL words from $\{0,1\}^k$ and $1.23n$ offset values in $[\sqrt{n}]$ (about $0.62n \log n$ bits).

5 Below the Graph Thresholds

Using the approach of Majewski *et al.* [22] one may not get below the space bound $1.23n'$ given by the requirement that the random hypergraphs be acyclic. The Undo-Une construction from [13] achieves space $(1+\varepsilon)n'$, but seemingly not less. However, in [4] and in [31] methods for constructing MPHF are described that have the potential to get below the threshold of n' words. Botelho *et al.* [4] as well as Weidling [31] independently propose using the hypergraph approach with $d = 2$, in which case the hypergraph $G(S', h_1, h_2)$ turns into a standard graph. The central change is to give up the requirement that this graph be acyclic. Rather, these authors propose studying the 2-*core* $J_2 \subseteq [m']$ of $G(S', h_1, h_2)$, which is the largest subgraph all of whose nodes having degree 2 or larger.

From graph theory it is well known that the 2-core of a graph G can be found in linear time by a simple "peeling" process. This process iterates cutting off nodes of degree 1 ("leaves") from G; the remaining graph with nodes of degree at least 2 is the 2-core. If one assumes that the 2-core J_2 of $G(S', h_1, h_2)$ has been determined and that values $g(i)$, $i \in J_2$, have been calculated such that

$$\begin{aligned}&\text{the mapping } x \mapsto (g(h_1(x)) + g(h_2(x))) \bmod n' \\ &\text{is one-to-one on the set } \{x \in S' \mid h_1(x), h_2(x) \in J_2\},\end{aligned} \tag{9}$$

then it is very easy to calculate values $g(i)$ for $i \in [m'] - J_2$ such that $x \mapsto (g(h_1(x)) + g(h_2(x))) \bmod n'$ is one-to-one on S'. (See Section 5.2.) We are left with the problem of finding a suitable g-labeling of the nodes in J_2. Here, the methods of [4] and [31] differ.

5.1 A Partly Heuristic Approach

Botelho *et al.* [4] propose a greedy strategy for determining the g-values inside J_2. This strategy assigns g-values to nodes in the order $i_1, \ldots, i_{|J_2|}$ of a breadth-first-search in the 2-core, where each value $g(i_t)$ is chosen so that it is bigger than $g(i_1), \ldots, g(i_{t-1})$ but minimal so as not to get into conflict with (9). The authors of [4] conjecture (Conjecture 1, [4, p. 496]) that if

$$\text{the 2-core of } G(S', h_1, h_2) \text{ has } \le n'/2 \text{ edges,} \tag{10}$$

then this greedy strategy succeeds in the sense that the largest g-value assigned is not larger than $n' - 1$. For this conjecture, experimental evidence is provided.

It remains to estimate the edge density n'/m' we can afford so that with high probability the 2-core has no more than $n'/2$ edges. Referring to results on the structure of 2-cores of random graphs, in particular to [26], in [4] the following rule is provided (valid for $n', m' \to \infty$): The number of edges in the 2-core is

$$(1 + o(1))(1 - T/d)^2 n',$$

where $d = 2n'/m'$ is the average degree of $G(S', h_1, h_2)$ and T is the unique solution in $(0, 1)$ of the equation $Te^{-T} = de^{-d}$. A simple numeric computation shows that the threshold value for d is approximately 1.736. This means that (for n', m' large) we can afford $d \approx 1.73$, or $m' \ge 1.152n'$, and may expect to have no more than $n'/2$ edges in the 2-core.

In [4] experimental evidence is given that the algorithm works well for this choice of m'. The authors further report that in experiments a variant of their algorithm (not insisting that the values $g(x_t)$ increase with t increasing) makes it possible to further decrease m', to some value $m'/n' \approx 0.93$, but not further.

5.2 An Analyzed Approach

We turn to Weidling's [31] analysis of the strategy based on the 2-core of $G(S', h_1, h_2)$. The lowest edge density we consider in the analysis is never larger than 1.1, meaning that always $m' > 0.9n'$. In this case the probability that $G(S', h_1, h_2)$ has an empty 3-core (i. e., $G(S', h_1, h_2)$ does not have a nonempty subgraph with minimum degree 3) is overwhelming. Further, with high probability the maximum degree of nodes in $G(S', h_1, h_2)$ is $O(\log(n'))$. Finally, with positive probability $G(S', h_1, h_2)$ does not have double edges. For simplicity from here on we assume that $G(S', h_1, h_2)$ satisfies these properties (otherwise this will turn out at some time of the execution of the algorithm, in which case we choose new hash functions h_1, h_2 for S'). The following assumption is crucial for the analysis of the first algorithm (cf. (10)):

$$\text{the 2-core of } G(S', h_1, h_2) \text{ has } \le (\tfrac{1}{2} - \varepsilon)n' \text{ edges.} \tag{11}$$

We "peel" $G(S', h_1, h_2)$, as follows: Let $G_1 = G(S', h_1, h_2)$. We disregard nodes of degree 0.
• Round $t = 1, \ldots, \ell_1$: We choose a node j_t of degree 1 in G_t and obtain G_{t+1} by removing j_t and the (unique) incident edge $\{h_1(x_t), h_2(x_t)\}$. — What remains is a graph G_{ℓ_1} with minimum degree 2, the 2-core.
Round $t = \ell_1 + 1, \ldots, \ell_2$: If all nodes in G_t have degree 2 or larger, we choose a node j_t of degree 2 and obtain G_{t+1} by removing j_t and its two incident edges $\{h_1(x_{t,1}), h_2(x_{t,1})\}$, $\{h_1(x_{t,2}), h_2(x_{t,2})\}$ from G_t. Otherwise we choose a node j_t of degree 1, and obtain G_t by removing it and the (unique) incident edge $\{h_1(x_t), h_2(x_t)\}$. This is continued until an empty graph results.

Now the g-values are assigned. Preliminarily, assign g-value 0 to all nodes $j \in [m']$. Let $H = \emptyset$ (the already assigned hash values). We proceed in the reverse order of the peeling process.
Round $t = \ell_2, \ldots, \ell_1 + 1$:

Case 1: Node j_t has degree 2 in G_t, with two incident edges $\{h_1(x_{t,1}), h_2(x_{t,1})\}$, $\{h_1(x_{t,2}), h_2(x_{t,2})\}$. Assume $j_t = h_1(x_{t,1}) = h_1(x_{t,2})$. (The other cases are treated analogously.) Let $j' = h_2(x_{t,1})$ and $j'' = h_2(x_{t,2})$. What are legal values for $g(j_t)$ so that we do not get stuck on our way to constructing a MPHF? We must have
(i) $(g(j_t) + g(j')) \bmod n'$, $(g(j_t) + g(j'')) \bmod n'$ are *different* and not in H, and
(ii) $g(j_t) \notin \{g(j_s) \mid s > t$ and j_s has distance 2 to $j_t\}$.
That condition (i) is necessary (and sufficient for carrying out step t) is obvious. Condition (ii) makes sure that in a later step t' it will not happen that a common neighbor of j_t and $j_{t'}$ cannot be labeled because $g(j_t) = g(j_{t'})$. Since by assumption (11) we have $|H| \leq \frac{1}{2}(1 - \varepsilon)n'$, and since graph $G(S', h_1, h_2)$ has maximum degree $O(\log(n'))$, conditions (i) and (ii) exclude at most $(1 - \varepsilon)n' - O(\log(n'))$ values for $g(j_t)$. Since there are n' values to choose from, we may try values from $[n']$ at random until a suitable value for $g(j_t)$ is found, and will succeed after an expected number of $1/\varepsilon$ rounds. One still has to prove the simple fact that the expected number of nodes at distance 1 and 2, averaged over all nodes, is $O(1)$, to conclude that the overall construction time is expected $O(n)$.

Case 2: Node j_t has degree 1 in G_t. Let j' be its unique neighbor in G_t. In this case the new value $g(j_t)$ just has to satisfy (i)$'$ $(g(j_t) + g(j')) \bmod n' \notin H$ and condition (ii); again after an expected constant number of random trials we will find a suitable value $g(j_t)$.
• Round $t = \ell_1, \ldots, 1$:
We know that node j_t has degree 1 in G_t. Let j' be its unique neighbor. Deterministically choose $g(j_t) = (i + n' - g(j')) \bmod n'$ for one (the next) element $i \notin H$.

This algorithm finishes in expected linear time, if (11) is satisfied. On the same grounds from random graph theory as noted in Section 5.1 one sees that for this the condition $m' \geq 1.152(1 + \delta)n'$ for some $\delta > 0$ is sufficient. (In [31] a direct estimate of the number of edges in the 2-core is provided, leading to the same result.)

Looking from the point of view of space efficiency, the threshold $m' \geq 1.152$ $(1+\delta)n'$ is not yet satisfying since the construction from [13] achieves a similar result with space $m' = (1+\delta)n'$.

Weidling [31] investigated the combination of the graph-based construction with the "Undo-One" strategy from Section 3.2. He proved that $m' \geq 0.9353(1+\delta)n'$ is sufficient to guarantee that an adapted version of this strategy succeeds in building a graph-based MPHF. For the full data structure in the context of the split-and-share approach this would lead to the same space requirements as in Section 4.1, replacing the term "$1.23n$ offset values in $[\sqrt{n}]$ (about $0.62n \log n$ bits)" by "$0.94n$ offset values in $[\sqrt{n}]$ (about $0.47n \log n$ bits)".

6 Below $n \log n$

Using methods different from those described in this paper, based on the approach of [20], Woelfel [32] provided a MPHF construction that had more practical evaluation times than the purely theoretical constructions but gets by with space $O(n \log\log n)$. A similar result was reported in [11]. This construction is based on the hash-and-displace approach with the random-fit-decreasing algorithm, see 3.1. However, for each bucket B_i of size 2 or larger a new sequence of fully random hash functions is employed (instead of one fixed f for all buckets). Only the index of the successful hash function has to be stored, which will be a number of size $O(\log n)$, hence of $\log\log n + O(1)$ bits. The buckets B_i of size 1 cause a new subtle problem. To allocate these buckets with a hash function out of a pool of $O(\log n)$ many, one has to construct a perfect matching in the graph induced by the x's in such buckets and the respective hash values $h_0, \ldots, h_{L-1}$, for $L = \Theta(\log n)$. For this, methods for finding matchings in sparse random graphs are employed ([1,24]). The construction time rises to $O(n(\log n)^2)$.

7 An Almost Optimal Solution

Very recently, Botelho, Pagh, and Ziviani ([5], WADS'07) described a method to obtain a MPFH with description size $O(n)$, a constant factor away from the optimum. We give a brief account of their approach. For the theoretical analysis, they appeal to the "split-and-share" approach just as we did before, so we may assume that we have to achieve the goal formulated at the end of Section 1.3: find a MPHF for S', $n' = |S'| \leq \sqrt{n}$, assuming a pool of $K \log n$ fully random functions. Botelho *et al.* set out from the hypergraph setting of Section 4.1, with hypergraphs of order d. They use the fact known from random graph theory that for each $d \geq 2$ there is a constant c_d such that if $m'/n' \geq c > c_d$ then the hypergraph $G(S', h_1, \ldots, h_d)$ is acyclic (with positive probability for $d = 2$ and with high probability as $n', m' \to \infty$ for $d \geq 3$). They calculate the corresponding order $(x_1, \ldots, x_{n'})$ of the elements of S' such that (8) is satisfied. The crucial observation now is that the mapping

$$x_j \mapsto \text{ some element } \ell_j \text{ of } e_{x_j} - \textstyle\bigcup_{s<j} e_{x_s}$$

from S' to $[m']$ is one-to-one. Thus, for each j we may choose some $\ell_j \in [d]$ (namely, an arbitrary $\ell_j \in e_{x_j} - \bigcup_{s<j} e_{x_s}$) such that the mapping

$$h'\colon U \ni x \mapsto h_{\ell_j+1}(x) \in [m'] \tag{12}$$

is one-to-one on S'. Thus this mapping already represents a hash function that is perfect on S', with a range that is a little larger than we would like it to be.

A central idea of [5] now is to provide a data structure for calculating h'. (As mentioned in Remark 1, this idea was already used in very much the same way by Chazelle *et al.* [7] in the context of data structures that represent "half-dynamic" mappings from S' to some range X, there called "Bloomier filters", but without an attempt to make m' as small as possible.) This is done as follows: We use the construction indicated in Lemma 3, but not for $[n']$ with modular addition, but for $[d]$: We may find, in linear time, values $g(i) \in [d]$, $i \in [m']$, such that

$$(g(h_1(x_j)) + \cdots + g(h_d(x_j))) \bmod d = \ell_j \text{ , for } 1 \le j \le n'.$$

Moreover, we arrange that $g(i) = 0$ for $i \notin \{\ell_j \mid 1 \le j \le n'\}$. These values $g(i)$, when stored in a table with m' entries from $[d]$, form a data structure that makes it possible to calculate $h'(x)$, $x \in U$, from (12) as follows:

$$h'(x) = h_{1+(g(h_1(x))+\cdots+g(h_d(x))) \bmod d}(x). \tag{13}$$

Storing the values $g(j), j \in [m']$, takes about m' blocks of $\lceil \log d \rceil$ bits, or, by coding s numbers into one block of length $\lceil \log(d^s) \rceil$, about m'/s blocks of $\lceil \log(d^s) \rceil$ bits. For a concrete figure, let $d = 3$, in which case one may use $m' = 1.23n'$, and $s = 5$. Then $\lceil \log(d^s) \rceil = \lceil \log(243) \rceil = 8$, so a block is a byte, and one needs $1.23n'$ bytes or approximately $1.97n'$ bits. Using this approach for constructing a perfect hash function for S one obtains a data structure that uses no more than $2n + O(n^{2/3} \log n)$ bits, and has an extremely simple structure.

The hash function h' described so far does not have minimal range $[n']$. In [5] the following approach is proposed. In the table for the g-values positions $j \in [m'] - \{\ell_j \mid 1 \le j \le n'\}$ are filled with the entry "d" — indicating that the index does not belong to the set $\{\ell_j \mid 1 \le j \le n'\}$, but having no effect on the arithmetic modulo d. Thus, for $d = 3$ one has four possible g-values, requiring 2 bits per entry, resulting in a little bit more than $2.46n'$ bits.

The MPFH h we aim at is defined as

$$h(x) = |\{\ell_j \mid 1 \le j \le n', \ell_j < h'(x)\}|. \tag{14}$$

A moment's thought reveals that this function h takes on values in $[n']$ and is one-to-one on S', because the $h'(x)$-values for the elements $x \in S'$ are just the n' elements in $\{\ell_j \mid 1 \le j \le n'\}$. There are several ways of calculating $h(x)$ using the table of g-values and some auxiliary data structure — for details see [5].

The authors of [5] report on experiments that indicate that their approach leads to data structures that are space-efficient as described by the theory as well as very time-efficient. (The implementations use standard universal hash classes, which turn out to be sufficient so that it is not necessary to employ the "split-and-share" trick.)

8 Conclusion

The study of data structures for the MPHF problem has taken some interesting steps in the past few years. From the theoretical side, it has been understood how the full randomness assumption may be justified without resorting to constructions with large evaluation times, and — building on earlier work that have demonstrated the crucial role played by hypergraph structures — a practically useful construction has been found that approaches the theoretically optimal space bound of $n \log e$ bits up to a small constant factor. A natural question left open is whether one can get even closer to the space lower bound, while retaining practicability. Also, it would be interesting to see whether provable randomness properties remain in the graph and hypergraph structures discussed here if one does not assume full randomness but only k-wise independence for some k.

References

1. Bast, H., Mehlhorn, K., Schäfer, G., Tamaki, H.: Matching algorithms are fast in sparse random graphs. Theory Comput. Syst. 39(1), 3–14 (2006)
2. Bentley, J.: Programming pearls: a sample of brilliance. J. Assoc. Comput. Mach. 30(9), 754–757 (1987)
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)
4. Botelho, F.C., Kohayakawa, Y., Ziviani, N.: A practical minimal perfect hashing method. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 488–500. Springer, Heidelberg (2005)
5. Botelho, F.C., Pagh, R., Ziviani, N.: Simple and space-efficient minimal perfect hash functions. In: Proc. 10th Workshop on Algorithms and Data Structures (WADS 2007). LNCS, vol. 4619, Springer, Heidelberg (to appear 2007)
6. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. 18(2), 143–154 (1979)
7. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The Bloomier filter: an efficient data structure for static support lookup tables. In: Proc. 15th ACM-SIAM SODA 2004, pp. 30–39 (2004)
8. Czech, Z.J., Havas, G., Majewski, B.S.: An optimal algorithm for generating minimal perfect hash functions. Inform. Proc. Lett. 43(5), 257–264 (1992)
9. Czech, Z.J., Havas, G., Majewski, B.S.: Perfect Hashing (Fundamental Study). Theor. Comput. Sci. 182(1–2), 1–143 (1997)
10. Demaine, E.D., Meyer auf der Heide, F., Pagh, R., Patrascu, M.: De dictionariis dynamicis pauco spatio utentibus (lat. On dynamic dictionaries using little space). In: Correa, J.R., Hevia, A., Kiwi, M.A. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 349–361. Springer, Heidelberg (2006)
11. Dietzel, L.: Speicherplatzeffiziente perfekte Hashfunktionen, Diplomarbeit, Technische Universität Ilmenau, Fakultät IA (in German) (2005)
12. Dietzfelbinger, M., Gil, J., Matias, Y., Pippenger, N.: Polynomial hash functions are reliable. In: Kuich, W. (ed.) Automata, Languages and Programming. LNCS, vol. 623, pp. 235–246. Springer, Heidelberg (1992)
13. Dietzfelbinger, M., Hagerup, T.: Simple minimal perfect hashing in less space. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 109–120. Springer, Heidelberg (2001)

14. Dietzfelbinger, M., Meyer auf der Heide, F.: Dynamic hashing in real time. In: Paterson, M.S. (ed.) Automata, Languages and Programming. LNCS, vol. 443, pp. 6–19. Springer, Heidelberg (1990)
15. Dietzfelbinger, M., Karlin, A., Mehlhorn, K., Meyer auf der Heide, F., Rohnert, H., Tarjan, R.: Dynamic perfect hashing: Upper and lower bounds. SIAM J. Comput. 23(4), 738–761 (1994)
16. Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. Theoret. Comput. Sci. 380(1–2), 47–68 (2007)
17. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. Theory Comput. Syst. 38(2), 229–248 (2005)
18. Fredman, M.L., Komlós, J.: On the size of separating systems and families of perfect hash functions. SIAM J. Alg. Disc. Meth. 5(1), 61–68 (1984)
19. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with 0(1) worst case access time. J. Assoc. Comput. Mach. 31(3), 538–544 (1984)
20. Hagerup, T., Tholey, T.: Efficient minimal perfect hashing in nearly minimal space, in: Proc. 18th STACS 2001, Springer LNCS 2010. In: Ferreira, A., Reichel, H. (eds.) STACS 2001. LNCS, vol. 2010, pp. 317–326. Springer, Heidelberg (2001)
21. Knuth, D.E.: The Art of Computer Programming, vol. 2: Seminumerical Algorithms, 2nd edn. Addison-Wesley, Reading (1981)
22. Majewski, B.S., Wormald, N.C., Havas, G., Czech, Z.J.: A family of perfect hashing methods. Computer J. 39(6), 547–554 (1996)
23. Mehlhorn, K.: Data Structures and Algorithms, Vol. 1: Sorting and Searching. Springer, Berlin (1984)
24. Motwani, R.: Average-case analysis of algorithms for matchings and related problems. J. Assoc. Comput. Mach. 41(6), 1329–1356 (1994)
25. Pagh, R.: Hash and displace: Efficient evaluation of minimal perfect hash functions. In: Dehne, F., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 49–54. Springer, Heidelberg (1999)
26. Pittel, B., Wormald, N.C.: Counting connected graphs inside-out. J. Comb. Theory, Ser. B 93(2), 127–172 (2005)
27. Sanders, P.: personal communication
28. Schmidt, J.P., Siegel, A.: The spatial complexity of oblivious k-probe hash functions. SIAM J. Comput. 19(5), 775–786 (1990)
29. Siegel, A.: On universal classes of extremely random constant-time hash functions. SIAM J. Comput. 33(3), 505–543 (2004)
30. Thorup, M.: Even strongly universal hashing is pretty fast. In: Proc. 11th ACM-SIAM SODA 2000, pp. 496–497 (2000)
31. Weidling, C.: Platzeffiziente Hashverfahren mit garantierter konstanter Zugriffszeit, Dissertation. Technische Universität Ilmenau (in German) Electronic version. (2004), `http://www.db-thueringen.de/servlets/DocumentServlet?id=2431`
32. Woelfel, P.: Maintaining external memory efficient hash tables. In: Díaz, J., Jansen, K., Rolim, J.D.P., Zwick, U. (eds.) APPROX 2006 and RANDOM 2006. LNCS, vol. 4110, pp. 508–519. Springer, Heidelberg (2006)