A New Practical Algorithm for the Construction of a Perfect Hash Function

M. Atici
International Computer Institute
University of Ege
Izmir, Turkey
atici@ube.ege.edu.tr

D. R. Stinson
Department of Combinatorics and Optimization
University of Waterloo
Waterloo Ontario, N2L 3G1, Canada
dstinson@cacr.math.uwaterloo.ca

R. Wei
Department of Combinatorics and Optimization
University of Waterloo
Waterloo Ontario, N2L 3G1, Canada
rwei@cacr.math.uwaterloo.ca

Abstract

A perfect hash function for a subset X of $\{0, 1, \cdots, n-1\}$ is an injection h from X into the set $\{0, 1, \cdots, m-1\}$. Perfect hash functions are useful for the compact storage and fast retrieval of frequently used objects. In this paper, we discuss some new practical algorithms for efficient construction of perfect hash functions, and we analyze their complexity and program size.

Keywords: perfect hash family, perfect hash function, program size, complexity.

1 Introduction

A *hash function* is a function $h : \{1, 2, \cdots, n\} \rightarrow \{1, 2, \cdots, m\}$. A hash function is said to be *perfect* on a subset X of $\{1, 2, \cdots, n\}$ if h is injective on X, i.e., if $h|_X$ is one-to-one. Perfect hash functions are useful for the compact storage and fast retrieval of frequently used data, such as reserved words in programming languages, command names in interactive systems, etc. Let $w = |X|$; then $w \leq m$. When $w = m$, the function h is called a *minimal perfect hash function*. Minimal perfect hash functions have applications in compilers, operating systems, language translation systems, hypertext, hypermedia, file managers, and information retrieval systems. For more information about perfect hash functions and minimal perfect hash functions, readers can consult the recent survey paper [3] and its references.

The purpose of this paper is to present some new practical algorithms for construction of a perfect hash function. The efficiency of the algorithm is measured in three ways. First is the amount of time required to find a hash function h which is perfect on a given subset X. Second is the time required to evaluate a given function h for a given $x \in X$. Third is the amount of memory required to store a description of the function h, i.e., the *program size*. The memory required will be the logarithm of the number of the possible perfect hash functions in the associated *perfect hash family*, as defined below.

Definition 1.1 *An* (n, m, w)-perfect hash family *is a finite set of hash functions* $\mathcal{F}$ *such that*

$$h : A \rightarrow B$$

for each $h \in \mathcal{F}$*, where* $|A| = n$ *and* $|B| = m$*, with the property that for any* $X \subseteq A$ *such that* $|X| = w$*, there exists at least one* $h \in \mathcal{F}$ *such that* $h|_X$ *is one-to-one.*

We use the notation PHF$(N; n, m, w)$ to denote an (n, m, w)-perfect hash family with $|\mathcal{F}| = N$. We can think of a PHF$(N; n, m, w)$ as an $N \times n$ array of m symbols, where each row of the array corresponds to one of the functions in the family. This array has the

property that, for any subset of w columns, there exists at least one row such that the entries in the w given columns of that row are distinct. We will use this representation in some small examples in the sequel.

Let $N(n, m, w)$ denote the smallest N such that a PHF$(N; n, m, w)$ exists. In [4], $N(n, m, w)$ is proved to be $\Theta(\log n)$. However, the proof of [4] is not constructive, and it seems difficult to give explicit constructions that are good asymptotically. Hence, it is interesting to find explicit constructions for PHFs. We use some constructions where N is a polynomial function of $\log n$ (for fixed m and w). Moreover, our constructions have the advantage that they are simple and easy to program.

Our goal is to obtain an algorithm for construction and evaluation of a (minimal) perfect hash function in which the complexity and program size are low, and which also works well in practice.

The rest of this paper is arranged as follows. In Section 2, we describe our constructions of PHFs, both direct and recursive. Then we give algorithms to realize these constructions in Section 3. We will analyze the efficiency of these algorithms in Section 4.

All logarithms in this paper are to the base 2.

2 Constructions

2.1 Direct Constructions

In this section, we give two direct constructions of perfect hash families. These are simple "base" PHFs which will be used as initial families in our main recursive construction.

The first construction is based on a finite affine plane (see [1, Corollary 3.2]). Let q be a prime power such that $q + 1 > \binom{w}{2}$. Consider the array having columns indexed by pairs $(x, y) \in F_q \times F_q$, and rows indexed by $F_q \cup \{-1\}$, where F_q is a finite field with q elements and $-1 \notin F_q$. The entry in row r and column (x, y) is $x \times r + y$ if $r \in F_q$, and x if $r = -1$. It is easy to see that any two different columns have precisely one row of conflict. Since $q+1 > \binom{w}{2}$, it follows that for any w different columns there will be a row that has w different entries in these w columns. Hence we have the following result.

Theorem 2.1 *Suppose q is a prime power and $q+1 > \binom{w}{2}$. Then there exists a PHF$(q+1; q^2, q, w)$.*

Example 2.1 Let $m = 3$ and $w = 2$. In this case we can take $q = 3$, and we construct the following PHF$(4; 9, 3, 2)$, $\mathcal{F}$:

r	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)
-1	0	0	0	1	1	1	2	2	2
0	0	1	2	0	1	2	0	1	2
1	0	1	2	1	2	0	2	0	1
2	0	1	2	2	0	1	1	2	0

For any $w = 2$ columns of $\mathcal{F}$ there exist a row where two entries are different.

In the first construction, m is $\Omega(w^2)$. We will use another base PHF when $m \leq \binom{w}{2}$. This construction is far from optimal, however, the hash families are very easy to compute, and they are considerably smaller than the trivial construction in which $N = \binom{n}{w}$.

For a subset $X = \{x_1, x_2, \cdots, x_w\}$, where $x_1 < x_2 < \cdots < x_w$, we define a perfect hash function h_X as follows:

$$h_X(x) = \begin{cases} 1 & \text{if } x < x_2 \\ 2i-1 & \text{if } x_{2(i-1)} < x < x_{2i}, \text{ for some } i = 2, 3, \cdots, \lfloor \frac{w}{2} \rfloor \\ 2i & \text{if } x = x_{2i} \text{ for some } i = 1, 2, \cdots, \lfloor \frac{w}{2} \rfloor \\ w & \text{if } x \geq x_w. \end{cases}$$

Since $h_X(x_i) = i$ for $i = 1, 2, \cdots, w$, h_X is perfect on X. Thus the family

$$\{h_X : X \subseteq \{1, 2, \cdots, n\}, |X| = w\}$$

is a PHF. (In fact, it is a minimal PHF.)

Let us determine the number of functions, N, in the family. Observe that h_X depends only on the values $x_2, x_4, \cdots$. First, consider the case where w is even. Denote $t = w/2$. We have that $\{x_2, x_4, \cdots, x_{2t}\} \subseteq \{2, \cdots, n\}$ and $x_{2i} \geq x_{2(i-1)} + 2$. If we define $d_i = x_{2i} - i$ for $1 \leq i \leq t$, then we construct a list of t distinct elements $d_1, \ldots, d_t$ where $\{d_1, \ldots, d_t\} \subseteq \{1, \ldots, n-t\}$. Thus we see that

$$N = \binom{n - \frac{w}{2}}{\frac{w}{2}}$$

in the case where w is even. A similar argument when w is odd establishes the following theorem.

Theorem 2.2 *For any $n > w$, the PHF$(N; n, w, w)$ constructed above has*

$$N = \binom{n - \lceil \frac{w}{2} \rceil}{\lfloor \frac{w}{2} \rfloor}.$$

Example 2.2 Let $m = w = 3$. We construct a PHF$(\binom{3}{1}; 5, 3, 3)$, $\mathcal{F}$, as follows:

$$\mathcal{F} = \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 3 & 4 \\ \hline 1 & 2 & 3 & 3 & 3 \\ 1 & 1 & 2 & 3 & 3 \\ 1 & 1 & 1 & 2 & 3 \\ \hline \end{array}$$

For any 3-element subset X of $\{0, 1, 2, 3, 4\}$, there exists a row that has different entries in the corresponding columns from set X.

We mentioned that this construction represents a considerable improvement over the trivial family. For example, if $n = 11$ and $w = 5$, then we have $N = \binom{8}{2} = 28$ as compared to $N = \binom{11}{5} = 462$ in the trivial PHF.

In the case $n = w + 1$, we obtain a PHF$\big(\lfloor \frac{w}{2} \rfloor + 1; n, w, w\big)$, which can be shown to be optimal.

2.2 A Recursive Construction

In this section, we describe the recursive construction given in [1]. We begin with a specific type of difference matrix. Suppose that the integers n, w have the property that $\gcd(n, \binom{w}{2}!) = 1$. Let $D = (d_{i,j})$, where $d_{i,j} = ij \bmod n$ for $0 \leq i \leq \binom{w}{2}$ and $0 \leq j \leq n - 1$. This is called an $(n, \binom{w}{2} + 1)$-*difference matrix*, since for all i_1, i_2 such that $0 \leq i_1 < i_2 \leq \binom{w}{2}$, we have $\{(d_{i_1,j} - d_{i_2,j}) \bmod n : 0 \leq j \leq n - 1\} = \mathbb{Z}_n$. (See [2] for more information about difference matrices.) The following lemma ([1, Theorem 4.1]) gives a recursive construction for PHF that uses difference matrices.

Lemma 2.1 *Suppose there is an* $(n_0, \binom{w}{2}+1)$*-difference matrix and a PHF*$(N_0; n_0, m, w)$*. Then there is a PHF*$((\binom{w}{2}+1)N_0; n_0^2, m, w)$*.*

For completeness, we present an outline of the construction. Let A be a PHF$(N_0; n_0, m, w)$ and let $D = (d_{i,j})$ be an $(n_0, \binom{w}{2}+1)$-difference matrix. For $0 \le j \le n_0 - 1$, let A^j denote the array obtained from A by letting the permutation σ^j act on the columns of A, where $\sigma(i) = (i-1) \bmod n_0$. Now let

$$B = \begin{array}{|c|c|c|c|} \hline B_{0,0} & B_{0,1} & \cdots & B_{0,n_0-1} \\ \hline B_{1,0} & B_{1,1} & \cdots & B_{1,n_0-1} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline B_{\binom{w}{2},0} & B_{\binom{w}{2},1} & \cdots & B_{\binom{w}{2},n_0-1} \\ \hline \end{array}$$

where $B_{i,j} = A^{d_{i,j}}, 0 \le i \le \binom{w}{2}, 0 \le j \le n_0 - 1$. Then B is the desired PHF.

To illustrate the construction we give an example below.

Example 2.3 We construct a PHF$(12; 25, 3, 3)$. A PHF$(3; 5, 3, 3)$, denoted A, is as follows.

$$A = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 3 & 3 \\ \hline 1 & 1 & 2 & 3 & 3 \\ \hline 1 & 1 & 1 & 2 & 3 \\ \hline \end{array}$$

Next, we construct the $(5, 4)$ difference matrix D:

$$D = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 2 & 4 & 1 & 3 \\ \hline 0 & 3 & 1 & 4 & 2 \\ \hline \end{array}$$

Then we obtain a PHF$(4 \times 3; 5^2, 3, 3)$, denoted B, as follows.

$$B = \begin{array}{|c|c|c|c|c|} \hline A^0 & A^0 & A^0 & A^0 & A^0 \\ \hline A^0 & A^1 & A^2 & A^3 & A^4 \\ \hline A^0 & A^2 & A^4 & A^1 & A^3 \\ \hline A^0 & A^3 & A^1 & A^4 & A^2 \\ \hline \end{array}$$

That is,

$$B = \begin{array}{|cccccccccccccc|} \hline 1 & 2 & 3 & 3 & 3 & ... & ... & ... & 1 & 2 & 3 & 3 & 3 \\ 1 & 1 & 2 & 3 & 3 & ... & ... & ... & 1 & 1 & 2 & 3 & 3 \\ 1 & 1 & 1 & 2 & 3 & ... & ... & ... & 1 & 1 & 1 & 2 & 3 \\ \hline 1 & 2 & 3 & 3 & 3 & ... & ... & ... & 3 & 1 & 2 & 3 & 3 \\ 1 & 1 & 2 & 3 & 3 & ... & ... & ... & 3 & 1 & 1 & 2 & 3 \\ 1 & 1 & 1 & 2 & 3 & ... & ... & ... & 3 & 1 & 1 & 1 & 2 \\ \hline 1 & 2 & 3 & 3 & 3 & ... & ... & ... & 3 & 3 & 1 & 2 & 3 \\ 1 & 1 & 2 & 3 & 3 & ... & ... & ... & 3 & 3 & 1 & 1 & 2 \\ 1 & 1 & 1 & 2 & 3 & ... & ... & ... & 2 & 3 & 1 & 1 & 1 \\ \hline 1 & 2 & 3 & 3 & 3 & ... & ... & ... & 3 & 3 & 3 & 1 & 2 \\ 1 & 1 & 2 & 3 & 3 & ... & ... & ... & 2 & 3 & 3 & 1 & 1 \\ 1 & 1 & 1 & 2 & 3 & ... & ... & ... & 1 & 2 & 3 & 1 & 1 \\ \hline \end{array}$$

where A^j denotes a cyclic shift of j columns to the left in the array A.

Using the difference matrices constructed above, and iterating Lemma 2.1, we have the following theorem.

Theorem 2.3 *[1] Suppose there exists a PHF$(N_0; n_0, m, w)$, where $gcd(n_0, \binom{w}{2}!) = 1$. Then there is a PHF$((\binom{w}{2}+1)^j N_0; {n_0}^{2^j}, m, w)$ for any integer $j \geq 1$.*

3 Algorithms for Construction of a Perfect Hash Function

In this section, we describe algorithms which realize the constructions of the previous section. The first two algorithms concern Theorem 2.1. Suppose q is a prime, $q+1 > \binom{w}{2}$, and $n \leq q^2$. For a given

Algorithm 3.1: Construction of a (q^2, q, w) hash function

Input:
q, w and $\{x_1, x_2, \ldots, x_w\} \subseteq \{0, 1, \cdots, q^2 - 1\}$,
where q is prime, $q + 1 > \binom{w}{2}$

(1) **for** $k := 1$ **to** w **do**
$i_k := \lfloor \frac{x_k}{q} \rfloor$,
$j_k := (x_k - i_k \times q) \bmod q$

(2) **If** all i_k's are different **then** $r := -1$
else find r, where $0 \leq r \leq q - 1$
such that $(i_k \times r + j_k) \bmod q$ are different for $k = 1, ..., w$

(3) the constructed hash function is h_r

w-subset X of $\{0, 1, \cdots, n - 1\}$, Algorithm 3.1 finds a hash function which is perfect on X and takes on values in $\{0, 1, \cdots, q - 1\}$, and outputs the description of that function. Algorithm 3.2 will use the description of the hash function to evaluate the function for any $x \in \{0, 1, \cdots, n - 1\}$.

Similarly, Algorithms 3.3 and 3.4 realize the construction of Theorem 2.2.

Finally, Algorithms 3.5 and 3.6 realize the recursive construction of Theorem 2.3. Suppose we have a "base" PHF$(N; n_0, m, w)$ and a w-subset $X \subseteq \{0, 1, \cdots, n_0^{2^j} - 1\}$, where $\gcd(n_0, \binom{w}{2}!) = 1$. Algorithm 3.5 finds the hash function, say h, which is perfect on X. Algorithm 3.6 evaluates the hash function h which is found by Algorithm 3.5 at any input $x \in \{0, 1, ..., n - 1\}$, where $n = n_0^{2^j}$. Notice that in Algorithm 3.5, we do not need to store the whole constructed PHF; we only need to store the base PHF.

It is straightforward to combine the previous algorithms to give a general construction. Suppose integers n, m, w, and a w-subset $X \subseteq \{0, 1, \cdots, n - 1\}$ are given. We want to find a hash function which is perfect on X. We can proceed as follows. First we find the smallest prime $q > \binom{w}{2}$, so it follows $\gcd(q, \binom{w}{2}!) = 1$. (Notice that this step is actually a preprocessing step, since it does

Algorithm 3.2: Evaluate a (q^2, q, w) hash function

Input:
q, w, $x \in \{0, 1, \cdots, q^2 - 1\}$ and $r \in \{-1, 0, \cdots, q - 1\}$,
where q is prime, $q + 1 > \binom{w}{2}$

(1) $z := x$

(2) $i := \lfloor \frac{z}{q} \rfloor$
$j := (z - q \times i) \bmod q$

(3) **If** $r = -1$ **then** $h_r(x) := i$
else $h_C(x) := (i \times r + j) \bmod q$

Algorithm 3.3: Construction of an (n, w, w) hash function

Input: n, w and $\{x_1, x_2, \cdots, x_w\} \subseteq \{0, 1, \cdots, n - 1\}$

(1) Sort the elements $x_1, x_2, \cdots, x_w$ such that $x_1 < x_2 < \cdots < x_w$

(2) Let $t := \lfloor \frac{w}{2} \rfloor$

(3) Define $c_i := x_{2i}$, for $1 \leq i \leq t$

(4) The constructed hash function is denoted as h_C, where
$C := (c_1, c_2, \cdots, c_t)$

Algorithm 3.4: Evaluate an (n, w, w) hash function

Input:
n, w, $x \in \{0, 1, \cdots, n-1\}$ and $C = (c_1, c_2, \cdots, c_t)$ $(t = \lfloor \frac{w}{2} \rfloor)$

(1) Define $c_0 := -1$ and $c_{t+1} := n$

(2) **For** $i := 1$ to $t + 1$ **do**

(a) **If** $x = c_i$ **then** define $y := 2i$ and exit loop

(b) **If** $c_{i-1} < x < c_i$ **then** define $y := 2i - 1$ and exit loop

(3) Define $h_C(x) := y$

Algorithm 3.5: Construct an (n, m, w) hash function, $n = n_0^{2^j}$

Input:
A base PHF$(N; n_0, m, w)$, where $\gcd(n_0, \binom{w}{2}!) = 1$
$X = \{y_1, ..., y_w\} \subseteq \{0, 1, ..., n_0^{2^j} - 1\}$
(1) While $j > 0$ **do**
(a)**for** $k := 1$ **to** w **do**

$$i_k{}^{(j)} := \left\lfloor \frac{y_k}{n_0^{2^{(j-1)}}} \right\rfloor$$

$$x_k{}^{(j)} := [y_k - n_0^{2^{(j-1)}} \times i_k{}^{(j)}] \bmod n_0^{2^{(j-1)}}$$

(b) find d_j, $0 \le d_j \le \binom{w}{2}$, such that $(x_k{}^{(j)} + d_j \times i_k{}^{(j)}) \bmod n_0^{2^{(j-1)}}$

are all different for $k = 1, 2, \cdots, w$.

(c) **for** $k := 1, 2, \cdots, w$ **do**

$$y_k := (x_k{}^{(j)} + d_j \times i_k{}^{(j)}) \bmod n_0^{2^{(j-1)}}$$

(d) $j := j - 1$
(2) Find the hash function h_C in the base PHF$(N; n_0, m, w)$
which is perfect on $\{y_1, ..., y_w\}$
(3) The constructed hash function is denoted h_D
where $D = (d_j, d_{j-1}, ..., d_1; C)$

Algorithm 3.6: Evaluate an (n, m, w) hash function, $n = n_0^{2^j}$

Input:
$0 \le x \le n_0^{2^j} - 1$
$D = (d_j, ..., d_1; C)$
(1) /* Reduce x to $y \in [0, n_0 - 1]$ by using d_j's */
While $j > 0$ **do**
$i^{(j)} := \left\lfloor \frac{x}{n_0^{2^{(j-1)}}} \right\rfloor$
$l^{(j)} := (x - n_0^{2^{(j-1)}} \times i^{(j)}) \bmod n_0^{2^{(j-1)}}$
$x := (l^{(j)} + d_j \times i^{(j)}) \bmod n_0^{2^{(j-1)}}$
$j := j - 1$
(2) $y := x$
(3) $h_D(x) := h_C(y)$ where $h_C(y)$ is in the base PHF$(N; n_0, m, w)$

not require knowing the particular set X that is to be hashed.) Then we use Algorithm 3.5, letting $j = \lceil \log(\log n) - \log(\log n_0) \rceil$, to find the suitable hash function. If $q \le m$, then step (2) will use a PHF$(q+1; q^2, q, w)$ from Algorithm 3.1 as the base PHF, otherwise it will use a PHF$(\binom{q-\lceil \frac{w}{2} \rceil}{\lfloor \frac{w}{2} \rfloor}; q, w, w)$ from Algorithm 3.3 as the base PHF. (In this second case, it is not necessary that q be prime.) In either case, Algorithm 3.5 will output the description of the hash function.

We provide two examples below to illustrate the procedure.

Example 3.1 Let $m = w = 5$ and $n = 10^5$. The smallest prime $q > \binom{5}{2}$ is $q = 11$, and $m = 5 < 11$. Therefore we use a PHF$(\binom{8}{2}; 11, 5, 5)$ as a base PHF(Algorithm 3.3). Suppose

$$X = \{41, 614, 9958, 11125, 99987\}.$$

We apply Algorithm 3.5 with

$$j = \lceil \log(\log n) - \log(\log n_0)) \rceil = \lceil \log(\log(10^5)) - \log(\log(11)) \rceil = 3.$$

Iteration 1 $j = 3$

$X = \{41, 614, 9958, 11125, 99987\}$,
$(i_1^3, x_1^3) = (0, 41)$,
$(i_2^3, x_2^3) = (0, 614)$,
$(i_3^3, x_3^3) = (0, 9958)$,
$(i_4^3, x_4^3) = (0, 11125)$,
$(i_5^3, x_5^3) = (6, 12141)$.

If $d_3 = 0$, then

$$(41 + 0 \times d_3) \bmod 11^4 = 41,$$
$$(614 + 0 \times d_3) \bmod 11^4 = 614,$$
$$(9958 + 0 \times d_3) \bmod 11^4 = 9958,$$
$$(11125 + 0 \times d_3) \bmod 11^4 = 11125,$$
$$(99987 + 6 \times d_3) \bmod 11^4 = 12141,$$

are all different.

Iteration 2 $j = 2$

$X = \{41, 614, 9958, 11125, 12141\}$,
$(i_1^2, x_1^2) = (0, 41)$,
$(i_2^2, x_2^2) = (5, 9)$,
$(i_3^2, x_3^2) = (82, 36)$,
$(i_4^2, x_4^2) = (91, 114)$,
$(i_5^2, x_5^2) = (100, 41)$.

If $d_2 = 1$, then

$$(41 + 0 \times d_2) \bmod 11^2 = 41,$$
$$(9 + 5 \times d_2) \bmod 11^2 = 14,$$
$$(36 + 82 \times d_2) \bmod 11^2 = 118,$$
$$(114 + 91 \times d_2) \bmod 11^2 = 84,$$
$$(41 + 100 \times d_2) \bmod 11^2 = 20,$$

are all different.

Iteration 3 $j = 1$

$X = \{41, 14, 118, 84, 20\}$,
$(i_1^1, x_1^1) = (3, 8)$,
$(i_2^1, x_2^1) = (1, 3)$,
$(i_3^1, x_3^1) = (10, 8)$,
$(i_4^1, x_4^1) = (7, 7)$,
$(i_5^1, x_5^1) = (1, 9)$.

If $d_1 = 1$, then

$$(8 + 3 \times d_1) \bmod 11 = 0,$$
$$(3 + 1 \times d_1) \bmod 11 = 4,$$
$$(8 + 10 \times d_1) \bmod 11 = 7,$$
$$(7 + 7 \times d_1) \bmod 11 = 3,$$
$$(9 + 1 \times d_1) \bmod 11 = 10,$$

are all different.

Iteration 4 $j = 0$

$X = \{0, 4, 7, 3, 10\}$.
Apply Algorithm 3.3 with $X = \{0, 3, 4, 7, 10\}$; then $C = (3, 7)$.

Therefore the constructed hash function is h_D where $D = (0, 1, 1; 3, 7)$.

Now let us run Algorithm 3.6 on some different input values between 0 and 99999.

- Let $x = 11125$. x is first reduced to 11125, then to 118, then to 3. By Algorithm 3.4, $h_C(3) = 2$, thus $h_D(11125) = 2$.
- Let $x = 1000$. x is first reduced to 1000, then to 40, then to 10. Hence by Algorithm 3.4, $h_D(1000) = h_C(10) = 5$.
- Let $x = 15$. x is reduced to 15, then to 15, then to 5. By Algorithm 3.4, $h_D(15) = h_C(5) = 3$.

□

Example 3.2 Let $m = 12$, $w = 5$, and $n = 10^5$. Let

$$X = \{41, 614, 9958, 11125, 99987\}.$$

The smallest prime $q > \binom{5}{2}$ is $q = 11$, and $11 < m = 12$. Therefore we use a PHF$(12; 11^2, 11, 5)$ as a base PHF (Algorithm 3.1).

We apply Algorithm 3.5 with

$$j = \lceil \log(\log(10^5)) - \log(\log(11^2)) \rceil = 2.$$

Iteration 1 $j = 2$

$X = \{41, 614, 9958, 11125, 99987\}$,
$(i_1^2, x_1^2) = (0, 41)$,
$(i_2^2, x_2^2) = (0, 614)$,
$(i_3^2, x_3^2) = (0, 9958)$,
$(i_4^2, x_4^2) = (0, 11125)$,
$(i_5^2, x_5^2) = (6, 12141)$.

If $d_2 = 0$, then

$(41 + 0 \times d_2) \bmod 11^4 = 41$,
$(614 + 0 \times d_2) \bmod 11^4 = 614$,
$(9958 + 0 \times d_2) \bmod 11^4 = 9958$,
$(11125 + 0 \times d_2) \bmod 11^4 = 11125$,
$(99987 + 6 \times d_2) \bmod 11^4 = 12141$,

are all different.

Iteration 2 $j = 1$

$X = \{41, 614, 9958, 11125, 12141\}$,
$(i_1^1, x_1^1) = (0, 41)$,
$(i_2^1, x_2^1) = (5, 9)$,
$(i_3^1, x_3^1) = (82, 36)$,
$(i_4^1, x_4^1) = (91, 114)$,
$(i_5^1, x_5^1) = (100, 41)$.

If $d_1 = 1$, then

$(41 + 0 \times d_2) \bmod 11^2 = 41$,
$(9 + 5 \times d_2) \bmod 11^2 = 14$,
$(36 + 82 \times d_2) \bmod 11^2 = 118$,
$(114 + 91 \times d_2) \bmod 11^2 = 84$,
$(41 + 100 \times d_2) \bmod 11^2 = 20$,

are all different.

Iteration 3 $j = 0$

$X = \{41, 14, 118, 84, 20\}$

Apply Algorithm 3.1: For each $y_k \in X$ we compute (i_k, j_k) accordingly as follows:

$$(3, 8),\ (1, 3),\ (10, 8),\ (7, 7),\ \text{and}\ (1, 9).$$

The following table gives the entries of the corresponding columns in $\mathrm{PHF}(12; 11^2, 11, 5)$.

r	$(3,8)$	$(1,3)$	$(10,8)$	$(7,7)$	$(1,9)$
-1	3	1	10	7	1
0	8	3	8	7	9
1	0	4	7	3	10
2	3	5	6	10	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Therefore we can take $r = 1$, and hence $D = (0, 1; 1)$.

Now let us run Algorithm 3.6 on some different input values between 0 and 99999.

- Let $x = 12345$. x is first reduced to itself, then to 105. Now $105 = (9, 6)$ and $r = 1$, therefore $h_D(12345) = h_C(105) = (1 \times 9 + 6) \bmod 11 = 4$ by Algorithm 3.2.
- Let $x = 11125$. x is first reduced to itself, then to 84. $84 = (7, 7)$ and $r = 1$, therefore $h_D(11125) = h_C(84) = (1 \times 7 + 7) \bmod 11 = 3$ by Algorithm 3.2.

4 Efficiency of the Algorithms

In this section, we look at the efficiency of our algorithms. We are interested both in the complexity of the algorithms as well as the running time of an actual implementation. We consider the case of minimal perfect hashing, i.e., $m = w$. In this case, we can take q to be the smallest integer such that $\gcd(q, \binom{w}{2}!) = 1$. The value of q is less than w^2 and it can be computed in time polynomial in $\log w$, using the Euclidean algorithm to compute greatest common divisors.

Let's first consider program size. The size of the hash family in Theorem 2.3 is $(\binom{w}{2} + 1)^j N_0$. Here we have that $j \leq \log\log n$ and

$$N_0 < \binom{w^2}{\frac{w}{2}} < (w^2)^{\frac{w}{2}} = w^w.$$

Thus the program size is bounded above by

$$j \log w^2 + \log N_0 < 2 \log w \log\log n + w \log w.$$

It is known that a minimal PHF has program size $\Omega(\log\log n + w)$ (see, e.g., [4, p. 129] or [3, p. 9]). So our construction is not much larger than an optimal one.

In the encoding of PHF that we use, we have a sequence at most $\log\log n$ d's (produced by Algorithm 3.5), where $0 \leq d_i \leq q - 1$ for each i. Thus each d requires at most $\log q < 2 \log w$ bits to encode it. The remaining $w \log w$ bits correspond to the list of $\lfloor \frac{w}{2} \rfloor$ c's produced by Algorithm 3.3, where $0 \leq c_i \leq w^2 - 1$ for each i.

The above analysis is a provable, worst-case bound. In practice, however, we usually do not require so much space. This is because the d's are frequently very small and may not require $2 \log w$ bits to encode them. It appears that the program size is more likely to be $O(w \log w + \log\log n)$, since $d_i = 0$ or 1 "most of the time". We will now try to justify this assumption with an informal heuristic argument and some experiments.

We have done some experiments for fixed w values to compute the average number of values that need to be tested to find an acceptable d in step 1(b) of Algorithm 3.5. In these computations, we take $n = 2^{30}$ and randomly create 100 subsets X ($|X| = w$) of $\{0, 1, ..., 2^{30} - 1\}$ for each w. We run the program for each subset X, computing the

Table 4.1

w	q	j	Aver. num. of d values tested	$j-1+e$	time (seconds)
3	5	4	4.89	5.72	0.00013
10	47	3	4.72	4.72	0.00033
20	191	2	3.72	3.72	0.00067
50	1229	2	4.13	3.72	0.00248
100	4951	2	3.25	3.72	0.00668
150	11177	2	3.60	3.72	0.01373
200	19913	2	3.92	3.72	0.02298
300	44851	1	2.60	2.72	0.04432
400	79801	1	2.44	2.72	0.07488
500	124753	1	2.81	2.72	0.11518
600	179717	1	2.76	2.72	0.16355
700	244667	1	2.82	2.72	0.21934
800	319601	1	2.68	2.72	0.28208
900	404557	1	2.28	2.72	0.35176
1000	499507	1	2.91	2.72	0.43724

average number of tested d values. In Table 4.1, columns 1, 2, and 3 record the values of w, q, and j, respectively. The fourth column gives average number of d values checked (over all j iterations). The sixth column gives the computing time in seconds. (We have done these computations with Sun Ultra 1 Model 140 in the International Computer Institute at the University of Ege, Turkey.)

We now derive a heuristic estimate for the number of d values tested (see column 5 of Table 4.1). If we choose k random elements from a set of n elements (repetition allowed), the probability that all k elements will be distinct is

$$p(k,n) = \left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{k-1}{n}\right) = \prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right). \quad (1)$$

In step 1(b) of iteration l of Algorithm 3.5, we find a value d such

that the w values $(x+d\times i) \bmod q^{2^{l-1}}$ are distinct elements in $Z_{q^{2^{l-1}}}$. If we assume that these w values are randomly distributed, we can estimate the probability that they are distinct using Eq. (1) with $k = w$ and $n = q^{2^{l-1}}$.

For $l > 1$, $p(w, q^{2^{l-1}}) \approx 1$ while for $l = 1$, we have

$$\begin{aligned} p(w,q) &\approx p(w, w^2/2) = \left(1 - \frac{1}{w^2/2}\right) \cdots \left(1 - \frac{w-1}{w^2/2}\right) \\ &\approx \prod_{i=1}^{w-1} e^{-(\frac{i}{w^2/2})} \approx e^{-1}. \end{aligned}$$

Here, we are using the approximation $1 - x \approx e^{-x}$, which is true when x is a small positive real number. Thus we estimate that the total number of d values tested should be approximately $j - 1 + e$. These estimates, recorded in column 5 of Table 4.1, are quite close to the actual experimental values obtained in column 4.

Now we briefly analyze the complexity of the algorithms to construct and evaluate a hash function. For simplicity, we use the "uniform cost" model which assumes that any arithmetic operation can be done in $O(1)$ time (see, e.g., [3, p.10]). As above, we are considering the case $w = m$ and we assume that $q < w^2$ has been determined in a preprocessing step.

Step (1) of Algorithm 3.5 requires $O(\log \log n)$ iterations. Each iteration takes time $O(w \log w)$ to test each particular d value (the w numbers in step 1(b) can be sorted to determine if they are distinct). The number of d values that need be considered is $O(w^2)$. Thus step (1) can be accomplished in time $O(w^3 \log w \log \log n)$. Algorithm 3.3 takes time $O(w \log w)$, so the total time is $O(w^3 \log w \log \log n)$. Notice also that, if the heuristic argument presented above is valid, then the time required to construct the hash function is reduced to $O(w \log w \log \log n)$.

For purposes of comparison, the deterministic algorithm presented in [4] requires time $O(w^3 \log w \log n)$, so our worst-case running time is better by a factor of $O(\log n / \log \log n)$. Also, the algorithm in [4] has $m = 3w$, whereas our algorithm allows minimal perfect hashing ($m = w$) to be accomplished.

The evaluation time of our algorithm is analyzed in a similar fashion. Algorithm 3.6 requires time $O(\log \log n)$, and Algorithm 3.4 can be modified to run in time $O(\log w)$ if a binary search is used. Therefore the total time is $O(\log w + \log \log n)$.

Finally, we should emphasize again that the algorithms are very suitable for practical use, as is shown by the timings in Table 4.1.

Acknowledgment

The research of the second and third authors was supported by NSF Grant CCR-9610138.

References

[1] M. Atici, S. S. Magliveras, D. R. Stinson and W.-D. Wei. Some Recursive Constructions for Perfect Hash Families. *Journal of Combinatorial Designs* **4** (1996), 353–363.

[2] C. J. Colbourn and J. H. Dinitz. *CRC Handbook of Combinatorial Designs*, CRC Press, 1996.

[3] Z. J. Czech, G. Havas and B. S. Majewski. Perfect Hashing, *Theoretical Computer Science* **182** (1997), 1–143.

[4] K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching*, Springer-Verlag, Berlin, 1984.