Quasi-perfect Hashing

ZBIGNIEW J. CZECH

Institute of Computer Science, Silesia University, 41–200 Sosnowiec, Poland
Email: zjc@silesia.pl

The idea of quasi-perfect hashing is introduced and applied to solve the static dictionary problem. Given a universe U and a set S of n distinct keys belonging to U, we propose a quasi-perfect hash function which allows one to find a key from S, stored in the hash table of size m, $m \geq n$, in $O(1)$ time. While looking up a key at most two probes in the hash table are made. Our main motivation is to minimize the memory requirement for representing the hashing scheme, retaining a high probability of finding quasi-perfect hash functions for arbitrary sets S. If we compare the method of quasi-perfect hashing to Fredman, Komlós and Szemerédi's two-level hashing for the bounded universe U, we find that it is superior with regard to both space and speed.

Received May 4, 1998; revised October 1, 1998

1. INTRODUCTION

Given a set S of n distinct keys belonging to a universe $U = \{1, 2, \ldots, u-1\}$, we would like to store the keys of S in some data structure so that the membership queries of the form '*Is x in S?*' can be answered quickly. This searching problem, also called the *dictionary problem*, is ubiquitous in computer science applications. If no deletion and insertion of elements in S occurs, then the dictionary problem is called *static*. Perfect hashing is one of the best methods to solve this problem. An overview of perfect hashing is given in [1, 2] and the area is surveyed in [3].

There are several approaches to construct perfect hash functions (PHF). Fredman, Komlós and Szemerédi (FKS) [4] proposed a two-level hashing scheme based on segmentation. Although the FKS scheme can be constructed for arbitrary n and u, its memory requirement is relatively high.

In this paper we propose quasi-perfect hashing as a novel approach to solve the static dictionary problem. Our main motivation is to minimize the memory requirement for representing the hashing scheme, retaining a high probability of finding quasi-perfect hash functions for arbitrary sets S. If we compare the method of quasi-perfect hashing to the FKS approach for the bounded universe U, we find that it is superior with regard to both space and speed.

The rest of the paper is organized as follows. Section 2 contains the basic definitions regarding perfect hashing. In Section 3 we present one of the approaches to construct perfect hash functions called segmentation. Section 4 introduces the idea of quasi-perfect hashing. In Section 5 we discuss the problem of finding quasi-perfect hash functions. Section 6 concludes the work.

2. PERFECT HASHING

Let $U = \{1, 2, \ldots, u-1\}$ be the universe for some positive integer u. For convenience we assume that u is a prime number. Let S be a set of n distinct elements, or keys, belonging to U. A hash function is a function $h : U \to M$ that maps the keys from S into some given interval of integers M, say $[0, m-1]$. Given a key $x \in S$, the hash function computes an address, i.e. an integer in $[0, m-1]$, for the storage or retrieval of x. The storage area used to store keys is known as a hash table.

Keys for which the same address is computed are called synonyms. Due to the existence of synonyms, a situation called collision may arise, in which two different keys have the same address. A perfect, or 1-probe, hash function (PHF) for S is an injection $h : U \to [0, m-1]$, i.e. for all keys $x, y \in S$ such that $x \neq y$ we have $h(x) \neq h(y)$, which implies that $m \geq n$. If $m = n$ and h is perfect, then we say that h is a minimal perfect hash function (MPHF). It follows from the definition that a PHF transforms each key of S into a unique address in the hash table. Since no collisions occur, each key can be retrieved from the table in a single probe.

3. PERFECT HASHING WITH SEGMENTATION

An important approach to construct a PHF is segmentation. It divides an input set of keys into a number of subsets. For each subset a PHF is determined separately.

One of the methods of finding PHFs based on segmentation was proposed by Fredman *et al.* [4]. The method comprises two steps. First, given a set S of n distinct keys belonging to the universe U, a partition of S into n subsets B_i, $i = 0, 1, \ldots, n-1$, of size $b_i = |B_i|$ is obtained by using a primary hash function

$$h : x \mapsto ((ax) \bmod u) \bmod n$$

where $a \in [1, u-1]$ is a multiplier, and u is assumed to be prime. The subsets are also called collision buckets. Then for each collision bucket a secondary perfect hash function is constructed, of the form

$$h_i : x \mapsto ((a_i x) \bmod u) \bmod c_i$$

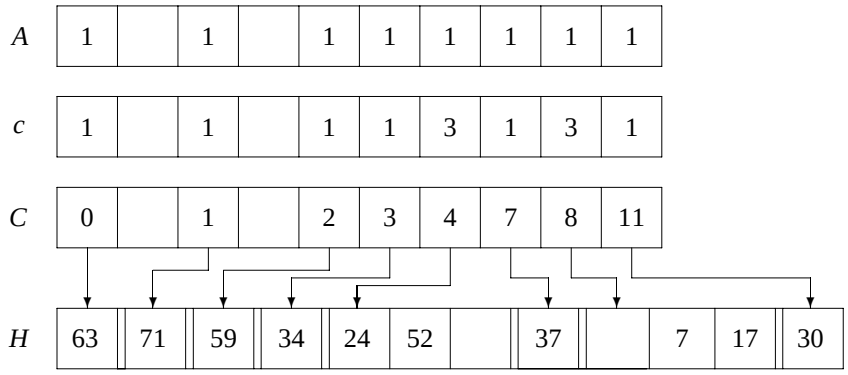

FIGURE 1. Representation of the PHF.

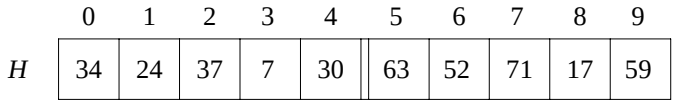

FIGURE 2. The hash table for the MQHF defined by Equation (1).

where $a_i \in [1, u-1]$ is a multiplier and $c_i = b_i^2$ is the size of the hash table for bucket B_i. Mehlhorn showed [5, Corollary III.2.3.10b] that the size of the hash table for bucket B_i can be reduced to $c_i = b_i(b_i - 1) + 1$.

EXAMPLE 1. Let $S = \{7, 17, 24, 30, 34, 37, 52, 59, 63, 71\}$, $n = 10$, $u = 101$, $a = 4$. The representation of the PHF for S consists of tables A, c and C indexed from 0 to 9, and table H indexed from 0 to 11 (Figure 1). Tables A and c contain the parameters a_i and c_i for the secondary hash functions. Table C contains the locations $C_i = \sum_{j=0}^{i-1} c_j$ which separate keys from different collision buckets in table H. The keys from S are stored in table H. A sample query for the key 17 proceeds as follows. The primary hash function gives index $i = ((4 \times 17) \bmod 101) \bmod 10 = 8$. Using this index we access tables A and c getting the parameters $a_8 = 1$ and $c_8 = 3$ for the secondary hash function, which gives $j = ((1 \times 17) \bmod 101) \bmod 3 = 2$. The key 17 is found in H at location $C_i + j = 8 + 2 = 10$.

Clearly, a membership query for a key $x \in S$ with the FKS scheme takes $O(1)$ time. Overall, a query involves two multiplications and four (integer) divisions of $(\log u)$-bit words to evaluate functions h and h_i, several accesses to parameters and array elements, and some additive operations. Lastly, one probe for key x in table H is made.

The compound hash function $h_i \circ h$ requires storing the parameters a, u and n for the primary hash function, the tables A and c containing the parameters a_i and c_i for the secondary perfect hash functions and the table C listing the offsets C_i in H. The description of the PHF needs $(3n + O(1)) \log u$ bits, plus $3n \log u$ bits for the keys in H, i.e. for a total $(6n + O(1)) \log u$ bits [4, Corollary 1].

Fredman *et al.* [4] also presented a modified hashing scheme which required $(n + o(n)) \log u$ bits. That scheme, however, is quite complicated and is of theoretical interest rather than suitable for practical applications.

4. QUASI-PERFECT HASHING

A quasi-perfect hash function (QHF) $h = (h_1(x), h_2(x))$ is comprised of a pair of hash functions such that every key $x \in S$, $|S| = n$, is found in the hash table either at address $h_1(x)$ or $h_2(x)$. Hence, at most two probes to locate any key in the hash table are needed. If the size of the hash table equals the number of keys, i.e. $m = n$, and h is quasi-perfect, then we say that h is a minimal quasi-perfect hash function (MQHF).

In this paper we propose a QHF $h = (h_1(x), h_2(x))$ where

$$h_1(x) = ((ax) \bmod u) \bmod r \quad \text{and} \quad h_2(x) = h_1(x) + r, \tag{1}$$

$a \in [1, u-1]$ is a multiplier and $r \geq \lceil n/2 \rceil$ is a parameter which determines the size of the hash table ($|H| = m = 2r$). Using the QHF above, a query for a key $x \in S$ in H is processed as follows:

$k := ((ax) \bmod u) \bmod r$;
if $(H[k] = x)$ **cor** $(H[k + r] = x)$ **then** found;
else not found;
end if;

The hash table H is divided into two equal parts. A key x is looked up first in the lower part of the table, $H[0 \ldots r-1]$, and then, if not found, in its upper part, $H[r \ldots 2r-1]$. If n is even and $r = \lceil n/2 \rceil$, then a QHF is minimal.

EXAMPLE 2. Consider the set S of keys from Example 1. The following function is the MQHF for S (Figure 2): $h = (h_1(x), h_2(x))$, $h_1(x) = ((4x) \bmod 101) \bmod 5$, $h_2(x) = h_1(x) + 5$.

Looking up the key 17 we first examine the cell $H[h_1(17)] = H[3] = 7$ and then the cell $H[h_1(17) + 5] = H[8] = 17$. Notice the simplicity of the MQHF representation as compared to the FKS scheme (Figure 1). The representation consists of the hash table H and the

	0	1	2	3	4	5	6	7	8	9
H	63	17	71	24	59	34	52	37	7	30

FIGURE 3. The hash table for the MQHF.

memory words to store the values of a, u and r. No tables A, c and C are required.

The evaluation of the QHF defined by Equation (1) takes $O(1)$ time and is slightly faster in comparison to the FKS function. It requires one multiplication, two (integer) divisions, perhaps one additive operation and several accesses to $(\log u)$-bit words. Also, at most two probes for a key in the hash table are required.

As we shall see, for the universe $U = \{1, 2, \ldots, 100\}$ and an arbitrary set of keys $S \subseteq U$, the hash table has at most $1.65n$ cells, with high probability. Thus the description of the QHF needs $(1.65n + O(1)) \log u$ bits, which is less than for the FKS hash function but still above the lower bound (equals $\Omega(n + \log\log u)$ bits, see [5]).

EXAMPLE 3. Another MQHF for S from Example 1 can be $h = (h_1(x), h_2(x))$ where $h_1(x) = ((4x) \bmod 101) \bmod 10$ and $h_2(x) = ((6x) \bmod 101) \bmod 10$ (Figure 3). Here $h_1(x)$ and $h_2(x)$ are entirely independent. QHFs of this form are not discussed in this paper.

5. FINDING QUASI-PERFECT HASH FUNCTIONS

The QHF given in Equation (1) is defined by the values of a, u and r. Before we present an algorithm to find these parameters for a given set of input keys, we prove two lemmas. Let D be a two-dimensional matrix of hash table indices defined as $D[a, x] = ((ax) \bmod u) \bmod r$, for some fixed r, and all possible multipliers a and keys x, i.e. for $a = 1, 2, \ldots, u-1$ and $x = 1, 2, \ldots, u-1$.

LEMMA 1. *For every pair (a, x) it holds that $D[a, x] = D[u-a, u-x]$, and the matrix D is symmetric with respect to both of its diagonals.*

Proof. The first part of the lemma follows from the equalities

$$((u-a)(u-x)) \bmod u = u^2 \bmod u - (u(a+x)) \bmod u + (ax) \bmod u = (ax) \bmod u.$$

The symmetry of matrix D with respect to its main diagonal is implied by the commutativity of multiplication, as we have

$$D[a, x] = ((ax) \bmod u) \bmod r = ((xa) \bmod u) \bmod r = D[x, a].$$

From the first part of the lemma and the symmetry of D with respect to its main diagonal we get

$$D[a, x] = D[u-a, u-x] = D[u-x, u-a]$$

which proves the symmetry of D with respect to its opposite diagonal. □

LEMMA 2. *Let $D[a, x_1] = D[a, x_2] = d$ for some keys $x_1 \neq x_2$. Then $D[u-a, x_1] = D[u-a, x_2] = (u-d) \bmod r$. In other words, if the keys x_1 and x_2 collide under the function $x \mapsto ((ax) \bmod u) \bmod r$ at address d, then they also collide under the function $x \mapsto (((u-a)x) \bmod u) \bmod r$ at address $(u-d) \bmod r$.*

Proof. From the assumption that the keys x_1 and x_2 collide under the function $x \mapsto ((ax) \bmod u) \bmod r$ it follows that $((ax_1) \bmod u) \bmod r = d$ and $((ax_2) \bmod u) \bmod r = d$ or equivalently

$$ax_1 - u\left\lfloor \frac{ax_1}{u} \right\rfloor = ir + d \tag{2}$$

$$ax_2 - u\left\lfloor \frac{ax_2}{u} \right\rfloor = jr + d \tag{3}$$

for some $i, j = 0, 1, 2, \ldots$. Subtracting these equations we get

$$ax_1 - u\left\lfloor \frac{ax_1}{u} \right\rfloor - ax_2 + u\left\lfloor \frac{ax_2}{u} \right\rfloor = (i-j)r. \tag{4}$$

Now assume that the keys x_1 and x_2 do *not* collide under the function $x \mapsto (((u-a)x) \bmod u) \bmod r$, that is

$$(((u-a)x_1) \bmod u) \bmod r = d_1,$$
$$(((u-a)x_2) \bmod u) \bmod r = d_2,$$

for some addresses d_1 and d_2, $d_1 \neq d_2$. These equations can be written as

$$-ax_1 - u\left\lfloor \frac{-ax_1}{u} \right\rfloor = kr + d_1,$$
$$-ax_2 - u\left\lfloor \frac{-ax_2}{u} \right\rfloor = lr + d_2,$$

where $k, l = 0, 1, 2, \ldots$. Using the formulae $\lfloor -x \rfloor = -\lceil x \rceil$ and $\lceil x \rceil = \lfloor x \rfloor + 1$ we obtain

$$-ax_1 + u\left\lfloor \frac{ax_1}{u} \right\rfloor + u = kr + d_1, \tag{5}$$

$$-ax_2 + u\left\lfloor \frac{ax_2}{u} \right\rfloor + u = lr + d_2. \tag{6}$$

Subtracting Equations (6) and (5), and using Equation (4) we have

$$(i-j)r = (l-k)r + d_2 - d_1. \tag{7}$$

Since $|d_2 - d_1| < r$, Equation (7) holds if and only if $d_1 = d_2$, which proves that the keys x_1 and x_2 *do* collide under the function $x \mapsto (((u-a)x) \bmod u) \bmod r$. From Equations (2) and (5) it follows that

$$-ir - d + u = kr + d_1$$

or

$$u - d = (k - i)r + d_1.$$

This means that the keys collide at address $d_1 = (u - d) \bmod r$. □

EXAMPLE 4. The address matrix D for the function $x \mapsto ((ax) \bmod 11) \bmod 5$ is shown in Figure 4. Note that the matrix is symmetric with respect to both of its diagonals. Consider the rows 3 and $11 - 3 = 8$. In compliance with Lemma 1 we have $D[3, 1] = D[8, 10]$, $D[3, 2] = D[8, 9], \ldots, D[3, 10] = D[8, 1]$. These equalities hold for any pair of rows $(i, u - i)$. Consider a sample set of keys, $\{2, 3, 6, 7, 8, 9\}$. Figure 5 shows the distributions of the keys in the hash table obtained by using the functions with $a = 3$ and $a = 8$. It can be seen that the patterns of these distributions are similar, i.e. the same keys collide in both distributions, though at different hash addresses (cf. Lemma 2).

Now we are prepared to present the algorithm for finding QHFs:

for $r := \lceil n/2 \rceil$ **to** $3n$ **do**
  for $a := 1$ **to** $(u - 1)/2$ **do**
    Empty the buckets B_i, $i = 0, 1, \ldots, r - 1$;
    Distribute the keys of the input
      set $S = \{x_1, x_2, \ldots, x_n\}$ over
      the buckets using the function
      $h_1 : U \to ((ax) \bmod u) \bmod r$;
    if $\max_{0 \le i \le r-1}\{|B_i|\} \le 2$ **then**
      {a and r are the parameters
        of the sought QHF}
      return(success);
    end if;
  end for;
end for;
return(failure);

The algorithm taking the set of keys $S = \{x_1, x_2, \ldots, x_n\}$ as input searches for the values a and r of the QHF defined by Equation (1). During the search the keys of the set S are repeatedly distributed over the buckets $B_i, i = 0, 1, \ldots, r - 1$. For this purpose the function $h_1 : U \to ((ax) \bmod u) \bmod r$ is applied (the value of u can be taken as a nearest prime greater than the maximum key in S). Given a fixed r, $r \in [\lceil n/2 \rceil, 3n]$, a suitable multiplier a from the range $[1, (u - 1)/2]$ is sought. As follows from Lemma 2, only half of the possible multipliers need to be checked. The QHF is found if a distribution has no buckets with more than two keys and we shall now refer to that as a 2-element distribution. Given a set S, the search is conducted for increasing values of r. Therefore, a QHF with a minimum size of the hash table is always found. The search is bounded by the value $r = 3n$, or by the size of the hash table $|H| = 2r = 6n$ which corresponds to the memory requirement of the FKS scheme. If for a given set S the algorithm does not find a suitable r within the range $[\lceil n/2 \rceil, 3n]$, one may use the FKS scheme which is then more space efficient.

	x									
a	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	0	1	2	3	4	0
2	2	4	1	3	0	1	3	0	1	4
3	3	1	4	1	4	2	0	2	0	3
4	4	3	1	0	4	2	1	0	3	2
5	0	0	4	4	3	3	2	2	1	1
6	1	1	2	2	3	3	4	4	0	0
7	2	3	0	1	2	4	0	1	3	4
8	3	0	2	0	2	4	1	4	1	3
9	4	1	0	3	1	0	3	1	4	2
10	0	4	3	2	1	0	4	3	2	1

FIGURE 4. Matrix D.

	hash addresses				
$a = 3$	0	1	2	3	4
	7	2	6		3
keys	9		8		

	hash addresses				
$a = 8$	0	1	2	3	4
	2	7	3		6
keys		9			8

FIGURE 5. Distributions of keys.

It is easily observed that the worst case time complexity of the algorithm is $O(n^2u)$. To speed up the algorithm we may begin the search from $r := 0.382u^{0.288}$ instead of $r := \lceil n/2 \rceil$. We found (using Theorem 1) that for that value of r a QHF exists for over 90% sets of random keys.

EXAMPLE 5. The algorithm described above was used to find a QHF for the set of 107 keys $S = \{3, 10, 12, \ldots, 2248, 2249, 2688\}$ which arose in some practical application. A few primes u greater than 2688 were used to find the QHF with the smallest possible hash table. Each run of the algorithm took a few seconds of computation on a medium class workstation. The following QHF for the hash table of size 166 was established: $h = (h_1(x), h_2(x))$, where $h_1(x) = ((535x) \bmod 2693) \bmod 83$ and $h_2(x) = h_1(x) + 83$.

Now let us investigate how well the hash function $h_1 : U \to ((ax) \bmod u) \bmod r$ distributes the keys in the universe $U = \{1, 2, \ldots, u - 1\}$ over the cells of the hash table of size r.

LEMMA 3. *Let d, $d \in [0, m - 1]$, be an arbitrary index into the hash table, and let a, $a \in [1, (u - 1)/2]$, be a fixed multiplier in the hash function h_1. If m does not divide $u - 1$, then the number of colliding keys in the universe U at address d equals either $l_1 = \lfloor (u - 1)/r \rfloor + 1$ or $l_1' = l_1 - 1$. The number of addresses with l_1 colliding keys equals $(u - 1) \bmod r$. If r divides $u - 1$, then the*

number of colliding keys is the same for all d's and equals $l_2 = (u-1)/r$.

Proof. Let $\{x_1, x_2, \ldots, x_l\}$, $x_i \in U$, be the set of keys colliding at address d. It is easy to see that the keys satisfy Equations (8) for $d \in [1, r-1]$, and Equations (9) for $d = 0$

$$\begin{aligned} (ax_1) \bmod u &= d \\ (ax_2) \bmod u &= d + r \\ &\ldots \\ (ax_l) \bmod u &= d + (l-1)r \end{aligned} \tag{8}$$

$$\begin{aligned} (ax_1) \bmod u &= r \\ (ax_2) \bmod u &= 2r \\ &\ldots \\ (ax_l) \bmod u &= lr \end{aligned} \tag{9}$$

where $d + (l-1)r \le u-1$ and $lr \le u-1$, respectively.

Case 1. $d \in [1, r-1]$. The inequality which holds in this case can be rewritten as

$$l \le \frac{u-1-d}{r} + 1. \tag{10}$$

If r does not divide $u-1$, then we have $u-1 = r\lfloor (u-1)/r \rfloor + (u-1) \bmod r$, which inserted into (10) gives

$$l \le \frac{r\lfloor (u-1)/r \rfloor + (u-1) \bmod r - d}{r} + 1. \tag{11}$$

Clearly, for addresses $d \in [1, (u-1) \bmod r]$, the maximum (integer) l satisfying inequality (11) equals $l_1 = \lfloor (u-1)/r \rfloor + 1$. If $d > (u-1) \bmod r$, then this value drops to $l_1' = \lfloor (u-1)/r \rfloor = l_1 - 1$. If r divides $u-1$, then the maximum l satisfying inequality (10) equals $l_2 = (u-1)/r$.

Case 2. $d = 0$. Here Equations (9) hold with $lr \le u-1$, or $l \le (u-1)/r$. Depending on whether r divides $u-1$ or not, we get $l_2 = (u-1)/r$ and $l_1' = \lfloor (u-1)/r \rfloor$, respectively. □

EXAMPLE 6. Figure 6a shows a sample distribution of the keys from the universe $U = \{1, 2, \ldots, 28\}$ obtained by using the hash function $x \mapsto ((5x) \bmod 29) \bmod 7$. Since $m = 7$ divides $u - 1 = 28$, the distribution has a 'perfect shape' of a rectangle. Figure 6b depicts a distribution of keys in the universe $U = \{1, 2, \ldots, 30\}$ obtained by using the hash function $x \mapsto ((11x) \bmod 31) \bmod 9$. This distribution has a 'typical shape' for the case where m does not divide $u-1$. The numbers of keys colliding at hash addresses in both distributions are in agreement with Lemma 3.

We now present a QHF existence theorem.

THEOREM 1. *Let* $S = \{x_1, x_2, \ldots, x_n\}$ *be a set of keys randomly chosen from the universe* U. *The probability of there existing a QHF for S defined by Equation (1), with a fixed* $r \ge \lceil n/2 \rceil$, *equals*

$$\tilde{p} = 1 - q^{(u-1)/2}, \tag{12}$$

(a)

0	1	2	3	4	5	6
10	3	9	2	5	1	4
13	6	12	15	8	11	7
23	16	22	18	21	14	17
26	19	25	28	24	27	20

(b)

0	1	2	3	4	5	6	7	8
25	11	1	14	2	19	5	22	8
27	13	3	16	4	21	7	24	10
29	15	28	18	6	23	9	26	12
	17	30	20					

FIGURE 6. Sample distributions of keys.

where

$$q = 1 - \left[\sum_{i=0}^{i_{\max}} \binom{r}{\alpha - i} \binom{\beta}{2}^{\alpha - i} \binom{r - (\alpha - i)}{\gamma + 2i} \beta^{\gamma + 2i} \right] \Big/ \binom{u-1}{n}, \tag{13}$$

$\alpha = \lfloor n/2 \rfloor$, $\beta = (u-1)/r$, $\gamma = n \bmod 2$ *and* $i_{\max} = \min(\alpha, r - \alpha - \gamma)$.

Proof. Let R, $|R| = \binom{u-1}{n}$, be the family of all subsets of n keys (n fixed) randomly chosen from the universe U. Given the function $h_1 : U \to ((ax) \bmod u) \bmod r$, $r \ge \lceil n/2 \rceil$, each subset is uniquely distributed over the hash table. According to Lemma 3, roughly $\beta = (u-1)/r$ keys from U are placed by the function $h_1 : U \to ((ax) \bmod u) \bmod r$, for given a, at each cell of the hash table of size r. As mentioned before, for the QHF in form (1), the hash table H of size $2r$ is divided into two equal parts. The QHF exists if its $h_1(x)$ gives either a 1- or 2-element distribution of keys from S over the lower part of table H. In the former case, all the keys from S are placed in the lower part of table H, while the upper part of H is empty. In the latter case, the excessive keys in the lower part of H are 'transferred' to its upper part by the function $h_2(x)$. The sum in Equation (13) counts a number of 1- and 2-element distributions among all $\binom{u-1}{n}$ possible distributions of the subsets in R. Thus the fraction in Equation (13) gives the probability of such a distribution existing (and consequently a QHF) for the keys from S and for the function $h_1 : U \to ((ax) \bmod u) \bmod r$ with fixed a. The value of q in Equation (13) determines the probability that for fixed a the QHF does not exist. Since $(u-1)/2$ of a's can be tried (cf. Lemma 2), the probability that a suitable a is found equals $\tilde{p} = 1 - q^{(u-1)/2}$. As to the sum in Equation (13), it counts the 1- and 2-element distributions beginning with those containing a maximum number of pairs of keys in the cells of the hash table ($i = 0$) and ending with the distributions containing a minimum number of pairs ($i = i_{\max}$). If $i_{\max} = \alpha$, then the last term of the sum gives the number of 1-element distributions. □

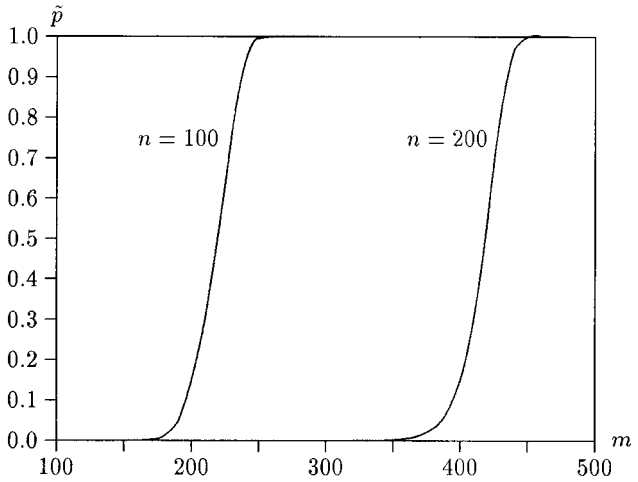

FIGURE 7. The probability of existing QHFs versus m ($m = 2r$) for $u = 503$.

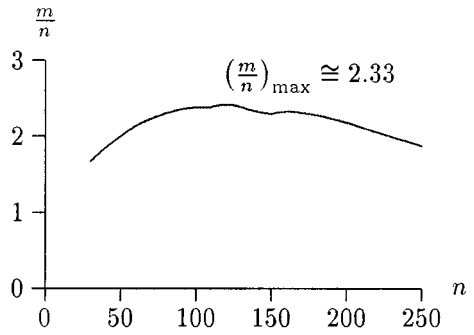

FIGURE 8. The ratio m/n versus n, for $u = 503$ and $Pr > 0.9$.

Using Theorem 1 we computed the probability $\tilde{p}$ of a QHF existing for the sets S of n keys randomly chosen from the universe $U = \{1, 2, \ldots, 502\}$ (Figure 7). It can be seen that for a given n the probability increases sharply when the size of the hash table reaches an appropriate value. For example, the QHFs can be found for over 50% of random sets of 200 keys if the size of the hash table equals 420. The rate of success exceeds 90% if that size equals 440. Figure 8 depicts the ratio $m/n = 2r/n$ versus n, for $u = 503$ and the probability Pr of there existing a QHF greater than 0.9. The ratio describes the space efficiency of a QHF, as it indicates how many cells in the hash table are needed to store n keys. Figure 8 shows that quasi-perfect hashing is more space efficient for sparse and dense sets S, i.e. the sets containing, respectively, a small and large part of the keys of the whole universe U. The general pattern of the graph in Figure 8, which exhibits a maximum for a certain cardinality of S, remains the same as the size of the universe U increases.

TABLE 1. Experimental results of finding QHFs for random sets

u	$(m/n)_{\text{ave}}$	σ
6007	2.94	0.63
7001	2.95	0.94
8009	3.22	0.74
9001	3.33	0.89
10,007	3.99	0.63
11,003	3.34	1.08
12,007	3.04	0.92
13,007	3.51	0.87
14,009	3.54	0.94
15,013	3.60	1.03

However the values of $(m/n)_{\max}$ also increase. We found that these values grow from $(m/n)_{\max} \cong 1.65$ for $u = 101$, to $(m/n)_{\max} \cong 6$ for $u = 15{,}013$.

In Table 1 the experimental results of finding QHFs for random sets of random size are shown. The columns of the table denote: u, the size of the universe from which the keys were taken (the sizes of key sets were random numbers from the range $[1, (u-1)/2]$); $(m/n)_{\text{ave}}$, the average value of the ratio m/n over 10 experiments; σ, the standard deviation of m/n values.

6. CONCLUSIONS

We introduced the idea of quasi-perfect hashing and applied it to solve the static dictionary problem. Our main motivation was to minimize the amount of space needed to build the hashing scheme. We showed that with probability $Pr > 0.9$ this amount does not exceed $(6n + O(1)) \log u$ bits, provided that the keys are taken from the universe of size $|U|_{\max} \cong 15 \times 10^3$. For such a bounded universe (which often occurs in practical applications) the method of quasi-perfect hashing is superior to the FKS approach with regards to both space and speed.

REFERENCES

[1] Gonnet, G. H. and Baeza-Yates, R. (1991) *Handbook of Algorithms and Data Structures*. Addison-Wesley, Reading, MA.

[2] Lewis, T. G. and Cook, C. R. (1988) Hashing for dynamic and static internal tables. *IEEE Computer*, **21**, 45–56.

[3] Czech, Z. J., Havas, G. and Majewski, B. S. (1997) Perfect hashing. *Theor. Comput. Sci.*, **182**, 1–143.

[4] Fredman, M. L., Komlós, J. and Szemerédi, E. (1984) Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, **31**, 3, 538–544.

[5] Mehlhorn, K. (1984) *Data Structures and Algorithms: Sorting and Searching*. Springer, Berlin.