BRICS

Basic Research in Computer Science

BRICS RS-97-17 P. B. Miltersen: Deterministic Dynamic Dictionaries

Error Correcting Codes, Perfect Hashing Circuits, and Deterministic Dynamic Dictionaries

Peter Bro Miltersen

BRICS Report Series RS-97-17

ISSN 0909-0878 June 1997

See back inner page for a list of recent BRICS Report Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/97/17/`

Error Correcting Codes, Perfect Hashing Circuits, and Deterministic Dynamic Dictionaries

Peter Bro Miltersen*

BRICS†, University of Aarhus.
Email: bromille@brics.dk

Abstract

We consider dictionaries of size n over the finite universe $U = \{0,1\}^w$ and introduce a new technique for their implementation: error correcting codes. The use of such codes makes it possible to replace the use of strong forms of hashing, such as universal hashing, with much weaker forms, such as clustering.

We use our approach to construct, for any $\epsilon > 0$, a deterministic solution to the dynamic dictionary problem using linear space, with worst case time $O(n^\epsilon)$ for insertions and deletions, and worst case time $O(1)$ for lookups. This is the first deterministic solution to the dynamic dictionary problem with linear space, constant query time, and non-trivial update time. In particular, we get a solution to the static dictionary problem with $O(n)$ space, worst case query time $O(1)$, and deterministic initialization time $O(n^{1+\epsilon})$. The best previous deterministic initialization time for such dictionaries, due to Andersson, is $O(n^{2+\epsilon})$. The model of computation for these bounds is a unit cost

*Supported by the ESPRIT Long Term Research Programme of the EU under project number 20244 (ALCOM-IT).

†Basic Research in Computer Science, Centre of the Danish National Research Foundation

RAM with word size w (i.e. matching the universe), and a standard instruction set. The constants in the big-O's are independent upon w. The solutions are weakly non-uniform in w, i.e. the code of the algorithm contains word sized constants, depending on w, which must be computed at compile-time, rather than at run-time, for the stated run-time bounds to hold.

An ingredient of our proofs, which may be interesting in its own right, is the following observation: A good error correcting code for a bit vector fitting into a word can be computed in $O(1)$ time on a RAM with unit cost multiplication.

As another application of our technique in a different model of computation, we give a new construction of perfect hashing circuits, improving a construction by Goldreich and Wigderson. In particular, we show that for any set $S \subseteq \{0,1\}^w$ of size n, there is a Boolean circuit C of size $O(w \log w)$ with w inputs and $2 \log n$ outputs so that the function defined by C is 1-1 on S. The best previous bound on the size of such a circuit was $O(w \log w \log\log w)$.

1 Introduction

The *static* dictionary problem is the following: Given a set of keys $S \subseteq \{0,1\}^w$ with $|S| = n$, construct a data structure using $O(n)$ registers, so that membership queries "Is x in S?" can be answered efficiently, and, if $x \in S$, some information associated with x can be retrieved. In the *dynamic* dictionary problem, we must also be able to support insertions and deletions of keys.

We consider solutions to the dictionary problem on a RAM with registers containing w bits, i.e. we assume that the word size matches the size of the universe (this is often known as the *trans-dichotomous* model [16]). The RAM operates on its registers with a standard instruction set of direct and indirect addressing, conditional jump, addition, subtraction, bitwise Boolean operations, shifts, and multiplication. All instructions are unit cost.

The seminal result of Fredman, Komlós, and Szemerédi [14] shows that in this model, it is possible to construct static dictionaries for sets of size n using $O(n)$ registers, so that queries can be answered in time $O(1)$. Their solution is based on universal hashing [6]. Since then, several variations of their scheme, all with constant query time, but with different additional desirable properties have appeared, all based on universal hashing [13, 7, 8, 9, 10, 20, 1].

In this paper we introduce an alternative to universal hashing for implementing linear space dictionaries with constant query time: Error correcting codes combined with clustering, a weak form of hashing introduced by Andersson *et al* [2], used to partition an input set into Hamming balls. In [2], a full solution to the dictionary problem was obtained by applying a special purpose hash function, a cluster buster, *after* the application of a clustering function. The central and very simple observation of this paper is that if we apply an error-correcting code to our keys *before* applying the clustering function, clustering works almost as well as universal hashing, and since clustering works in a more transparent way, it makes efficient construction of the involved structures easier.

We use our approach to show

Theorem 1 *For any $\epsilon > 0$, there is a deterministic solution to the dynamic dictionary problem using space $O(n)$, with worst case time $O(n^\epsilon)$ for insertions and deletions and worst case time $O(1)$ for queries. Here, n is the current size of the set of keys in the dictionary.*

We prove Theorem 1 by a simple dynamization of the following static structure:

Theorem 2 *For any $\epsilon > 0$, there is a solution to the static dictionary problem with domain $\{0,1\}^w$ using space $O(n)$, with deterministic worst case query time $O(1)$ and with deterministic worst case initialization time $O(n^{1+\epsilon})$.*

Previously, no linear space deterministic solution to the dictionary problem with constant query time and non-trivial update time has been given, but there has been a sequence of results on deterministically initializing static dictionaries. Fredman, Komlós, and Szemerédi [14] proved that their dictionary can be initialized deterministically in time $O(n^3 w)$. Raman [20] shows that by derandomizing multiplicative hashing [9] using conditional probabilities, this can be improved to $O(n^2 w)$. Finally, Andersson [1] shows that Raman's solution can be modified to give a time bound $O(n^{2+\epsilon})$, i.e., that the dependence on w can be removed. Interestingly, Andersson uses fusion trees [15], and this introduces a flaw, *weak non-uniformity* into the solution.

In the terminology of Ben-Amram and Galil [3], a trans-dichotomous algorithm is called *uniform* if the code runs within the stated bounds even if the word size of the machine is not known until "run-time", i.e., if the run-time code of the algorithm takes w as an input. It is called *weakly non-uniform* if the code of the algorithm contains descriptions of word sized constants, depending on w (such as masks), which must be computed at "compile-time" without charge, in order for the stated bounds to hold at run-time. Weak non-uniformity should hardly come as an esoteric concept for people used to real programming, where computation at compile-time is common-place. The constants of fusion trees are easily computed in time polynomial in w, so, informally speaking, Andersson's solution can be compiled in time polynomial in w.

Our static and dynamic dictionaries are also weakly non-uniform. We shall need an $O(w)$-bit constant a with the property that the binary representation of na contains $\Omega(w)$ blocks of ones for every $n \leq 2^w$ (this will ensure that $x \to ax$ is a good error-correcting code). Unfortunately, while we can show that such a constant exists, we don't know if it can be determined in time $w^{O(1)}$, so we don't know if we can compile our deterministic solution in time polynomial in the word size of the machine. Thus, the non-uniformity of our solution may seem somewhat more serious than the non-uniformity of Andersson's solution.

Fortunately, if the constant is just picked at random, it will work with high probability. Thus, for people not happy about calling a solution using non-constructive constants for deterministic, here is an alternative interpretation of the theorems: There is a *slightly randomized*, completely explicit and uniform, solution to the dynamic dictionary problem using linear space, with update time $O(n^\epsilon)$ and query time $O(1)$. This means that we select $O(1)$ words at random when we buy our machine or install our dictionary software, and pre-process using time $w^{O(1)}$. After this, we never make another random choice, and with probability $1-\epsilon$, we will never exceed the stated time bounds for any operation on any dictionary we may care to initialize and operate on in the future, even if the operations are given by an adversary who saw the random choice we made.

As another application of our technique in a different model of computation, we give a new construction of perfect hashing circuits, improving a construction by Goldreich and Wigderson [17]. They considered the following problem: Given an arbitrary subset $S \subseteq \{0,1\}^w$ of size 2^k, and a parameter

$m \geq k$, we want to construct a Boolean circuit $C : \{0,1\}^w \rightarrow \{0,1\}^m$, so that for all $x \neq y \in S$, $C(x) \neq C(y)$. Goldreich and Wigderson show the following bound:

Theorem 3 (Goldreich and Wigderson) *For every w, k, and $k \leq m \leq 2k$, and for every $S \subseteq \{0,1\}^w$ of size 2^k, there is a circuit of size $2^{2k-m}w^{O(1)}$ mapping S 1-1 to $\{0,1\}^m$.*

Goldreich and Wigderson show their bound to be tight, but only up to a polynomial in w. Here, we look more closely at the dependence upon w. Examining the dependence of their bounds on w, we find that for $m = 2k$, their construction gives at best a circuit of size $O(w \log w \log\log w)$. This is because their construction is based on universal hashing, and the best known circuits for universal hashing (with the weakest possible definition of universal) are based on Schönhage and Strassen's multiplication circuit which has the stated size. For $m < 2k$, the situation becomes worse: If the proofs of [17] are left unmodified, the dependence on w becomes cubic. This can be optimized somewhat, but since the circuits of [17] are based on circuits for w-wise independent hashfunctions, it is not obvious how to get an $o(w^2)$ bound.

Using error correcting codes with linear sized encoding circuits [12, 21] combined with clustering, we obtain the following improved bounds:

Theorem 4 *For every $S \subseteq \{0,1\}^w$ of size 2^k, the following circuits exist:*

- *For any $\epsilon > 0$, a circuit of size $O(w)$ mapping S 1-1 to $(2+\epsilon)k$ bits.*
- *A circuit of size $O(w + k \log k)$ mapping S 1-1 to $2k$ bits.*
- *A circuit of size $O(w + k2^{2k-m} + k \log k)$, mapping S 1-1 to m bits for $k \leq m \leq 2k$.*
- *A circuit of size $O(w + k2^k)$, mapping S 1-1 to k bits.*

The paper is organized as follows: In Section 2 we show how to do error correcting codes in constant time on a RAM with unit cost multiplication, a technique which we think may be interesting in its own right. In Section 3 we reprove a lemma about clustering functions from [2] in a stronger form. In Section 4, we show how to combine error correcting codes and clustering with the static dictionaries of Raman and Andersson to get the improved initialization time for static dictionaries. In Section 5 we show how to dynamize our solution. Raman's solution uses derandomized universal hashing, and thus, universal hashing is still present in our structure. In order to argue the (mainly philosophical) point that error correcting codes combined with clustering yields a true alternative to universal hashing, we present, in Section 6, an alternative implementation, replacing Raman's structure with the double displacement structure of Tarjan and Yao [24]. This solution uses a non-standard instruction set. Finally, in Section 7 we prove our results on perfect hashing circuits.

Notation

When x and y are bit strings of equal length, we denote by x AND y the bitwise Boolean AND of x and y. $x[i]$ denotes the i'th bit of x from the left. If $I = [i, j] = \{i, i+1, \ldots, j\}$ is an interval of bits, $x[I]$ denotes $x[i]x[i+1]\ldots x[j]$. We denote by w the word size of the machine. Words are considered bit strings of length w. The domain of w-bit bit strings $\{0,1\}^w$ and the domain $\{0, 1, \ldots, 2^w - 1\}$ of w-bit integers will be identified in the natural way. All log's are base 2.

2 Multiplicative error correcting codes

Here, we show how to do error correcting codes on words in constant time with multiplication. The similarity to multiplicative hashing by Dietzfelbinger *et al* [9] is obvious.

Let $e : \{0,1\}^w \to \{0,1\}^{kw}$. We say that e is an error correcting code with *rate* $1/k$ and *minimum distance* d, if, for all $x \neq y \in \{0,1\}^w$, the Hamming distance between $e(x)$ and $e(y)$ is at least d. It has *relative minimum distance* δ if it has minimum distance δkw.

Lemma 5 *Given w, there exists an integer a between 0 and $2^{(k+1)w} - 1$, so that $e_a : \{0, \ldots, 2^w - 1\} \to \{0, \ldots, 2^{kw} - 1\}$ defined by*

$$e_a(x) = (ax \bmod 2^{(k+1)w}) \operatorname{div} 2^w$$

is an error correcting code with rate $1/k$ and relative minimum distance at least δ for any $\delta < \frac{1}{2}$ satisfying $\frac{1}{k} \leq 1 - H(\delta) - \delta$. Here, H is the binary *entropy function $H(p) = -p \log p - (1-p)\log(1-p)$. Furthermore, if a is chosen at random, e_a is an error correcting code with rate $1/k$ and relative minimum distance at least δ with probability at least $1-\epsilon$ for any $\delta < \frac{1}{2}$ satisfying $\frac{1}{k} \leq 1 - H(\delta) - \delta - \frac{\log(1/\epsilon)}{kw}$.*

Proof Let $x, y \in \{0,1\}^w$ with $x > y$. Let a be a random integer between 0 and $2^{(k+1)w} - 1$. We shall bound the probability that $e_a(x)$ and $e_a(y)$ have Hamming distance strictly less than $d = \lceil \delta k w \rceil$. If this is the case, there exists $0 \leq s, t < 2^{kw}$ such that

- the sum of the Hamming weights of s and t is less than d,
- s and t have their 1-bits in disjoint positions,
- $e_a(x) = e_a(y) + s - t$.

For use in the discussion after the proof, we note that the binary representation of $s - t$ contains less than d blocks of ones.

Thus we have

$$(ax \bmod 2^{(k+1)w}) \operatorname{div} 2^w = (ay \bmod 2^{(k+1)w}) \operatorname{div} 2^w + s - t,$$

i.e., that

$$ax \bmod 2^{(k+1)w} = ay \bmod 2^{(k+1)w} + (s-t)2^w + u$$

for some u between $-2^w + 1$ and $2^w - 1$. That is,

$$a(x-y) \equiv (s-t)2^w + u \pmod{2^{(k+1)w}}.$$

Write $x - y$ as $z2^i$ for an odd z. We have

$$az2^i \equiv (s-t)2^w + u \pmod{2^{(k+1)w}}.$$

Thus we must have $u = u'2^i$ for some u' and

$$az \equiv (s-t)2^{w-i} + u' \pmod{2^{(k+1)w-i}}.$$

Since z is odd, it has an inverse z' modulo $2^{(k+1)w-i}$, so we have

$$a \equiv ((s-t)2^{w-i} + u')z' \pmod{2^{(k+1)w-i}}$$

for some s, t, u'. The number of ways to choose s, t, u' is $(\sum_{j=0}^{d-1} \binom{kw}{j} 2^j) 2^{w-i}$. Any value of a modulo $2^{(k+1)w-i}$ is equally likely. Thus, the probability that $e_a(x)$ and $e_a(y)$ have Hamming distance less than d is at most

$$(\sum_{j=0}^{d-1} \binom{kw}{j} 2^j) 2^{w-i} 2^{-(k+1)w+i} = (\sum_{j=0}^{d-1} \binom{kw}{j} 2^j) 2^{-kw}$$

.

This probability depends only on the value of $x-y$. There are only $2^w - 1$ possible values of $x - y$. So, plugging in $d = \lceil \delta kw \rceil$, the probability of small Hamming distance for *some* x, y is at most

$$P_{w,k,\delta} = (\sum_{j=0}^{\lceil \delta kw \rceil - 1} \binom{kw}{j} 2^j) 2^{-kw} (2^w - 1). \tag{1}$$

For $\delta < \frac{1}{2}$ the inequality $\sum_{j=0}^{\lfloor \delta n \rfloor} \binom{n}{j} \leq 2^{nH(\delta)}$ holds (see e.g. [18, page 310]). From this, we see that $P_{w,k.\delta}$ is smaller than a given value ϵ if $H(\delta)kw + \delta kw - (k-1)w < \log \epsilon$. ¿From this, the statement of the lemma follows. □

For the asymptotic behavior of the application to dictionaries, all we actually need is the following version of the lemma

Lemma 6 *For any $\epsilon > 0$, there is a $\delta > 0$, so that for all sufficiently large w, if a random $\lfloor (1/\delta - 1)w \rfloor$-bit integer a is chosen, with probability at least $1 - \epsilon$, $x \to ax$ defines an error correcting code mapping $\{0, \ldots, 2^w - 1\}$ to $\{0, \ldots, 2^{\lfloor w/\delta \rfloor} - 1\}$ with rate and relative minium distance at least δ.*

We will assume that such an a is available to our algorithm at run-time, making it possible for the algorithm to compute an error correcting code for a word in constant time. This is the weakly non-uniform part of our algorithm.

For practical considerations, the exact rate and minimum distance of the code are of course relevant. Here is an example: For $w = 16$, and $k = 4$, equation (1) guarantees us the existence of an 80-bit number a so that $x \to (ax \bmod 2^{80}) \text{ div } 2^{16}$ is an error correcting code with minimum distance 11, i.e. relative minimum distance 11/64, and that, in fact, more than a 0.39 fraction of the possible a's have this property. However, one can actually do better: By a computer search, we have verified that

$$
\begin{aligned}
a = {} & 0111000000011011000111001011111100011000 10 \\
& 1110001101010100010101000110010010010101
\end{aligned}
$$

is an 80-bit number leading to a code with minimum distance exactly 13. We haven't found any 80-bit numbers leading to a code with minimum distance 14.

We know no efficient algorithm for finding a and verifying that it has the right property. It would be nice to have a polynomial (in w) algorithm, as this would make the non-uniformity of our solution more like the non-uniformity of fusion trees, whose "magical constants" have this property. Therefore, a solution to the following problem would be interesting:

Problem: Given w, construct in time polynomial in w, an $O(w)$-bit number a, so the binary representation of na for all n between 1 and $2^w - 1$ contains at least $\Omega(w)$ blocks of ones.

The existence of such an a follows from the proof of lemma 5, and so does the fact that $x \to ax$ is a good error correcting code. We now just want to construct one deterministically in time polynomial in w. We conjecture that this is possible.

As a final remark, let us note that the correspondence between hashing and error correction is quite strong and that the similarity between multiplicative hashing and multiplicative error correction is no accident. Indeed, the following proposition holds.

Proposition 7 *Fix any pairwise independent family $\mathcal{H}$ of hash functions mapping $\{0,1\}^w \to \{0,1\}^{kw}$. Then, some member of the family is an error correcting code with rate $1/k$ and relative minimum distance at least δ for any $\delta < \frac{1}{2}$ satisfying $\frac{2}{k} \leq 1 - H(\delta)$. Also, a random member of the family is, with probability at least $1-\epsilon$, an error-correcting code with rate $1/k$ and relative minimum distance at least δ for any $\delta < \frac{1}{2}$ satisfying $\frac{2}{k} \leq 1 - H(\delta) - \frac{\log(1/\epsilon)}{kw}$.*

Proof That $\mathcal{H}$ is pairwise independent means that for fixed $x_1, x_2 \in \{0,1\}^w$ and $y_1, y_2 \in \{0,1\}^{kw}$, if $h \in H$ is chosen at random,

$$\Pr[h(x_1) = y_1 \wedge h(x_2) = y_2] = 1/2^{2kw}.$$

Let $d = \lceil \delta kw \rceil$. For any particular value of y, only $\sum_{j=0}^{d-1} \binom{kw}{j}$ vectors in $\{0,1\}^{kw}$ have Hamming distance less than d to y. Thus, for any x_1 and x_2, there are at most $2^{kw} \sum_{j=0}^{d-1} \binom{kw}{j}$ values of (y_1, y_2), so that $h(x_1) = y_1$ and

$h(x_2) = y_2$ would violate the distance requirement for $h(x_1)$ and $h(x_2)$. So, if h is chosen at random, the probability that the distance requirement for $h(x_1)$ and $h(x_2)$ is violated is at most $2^{-2kw}2^{kw}\sum_{j=0}^{d-1}\binom{kw}{j}$, and the probability that the distance requirement is violated for *some* pair is at most $2^{2w}2^{-2kw}2^{kw}\sum_{j=0}^{d-1}\binom{kw}{j}$. The statement of the lemma follows by a derivation similar to the one in Lemma 5. □

Thus, an alternative proof for showing that multiplication yields good error correcting codes is to combine the above proposition with Dietzfelbinger's result [11] that the family of functions of the form e_a is a pairwise independent family. If one is interested in a self-contained proof, one should note that Dietzfelbinger's proof of pairwise independence is much more complicated than the proof of Lemma 5. For concrete values of w and k, the concrete minimum distance guarantee we get from the proof of Proposition 7 is often worse than the one we get from Lemma 5 (for instance, taking the previous example of expanding 16 bits to 64 bits, the proof of Proposition 7 only guarantees us a value of a yielding a code with minimum distance 8). However, for small rates, Proposition 7 will give a better bound on the minimum distance than Lemma 5.

Proposition 7 shows that most members of a pairwise independent family must be error correcting codes. Pairwise independence being a generalization of universality, it is interesting to note that Bierbrauer *et al* [4] show a completely different connection between error correcting codes and universal families; namely that a good error correcting code can be used to derive a (nearly) universal family of hash functions on an exponentially smaller domain. For a nice survey of this correspondence and its applications, see Stinton [22].

3 Clustering

The following Lemma is proved in a weaker form in [2]. Here, what is important to us is the good dependence on n in the time bound.

Lemma 8 *Let S be a set of vectors in $\{0,1\}^w$ with the property that the Hamming distance between any two vectors in S is at least δw for some constant $\delta > 0$. Then, using time $O(wn \log n)$, we can find a word m of Hamming weight $O(\log n)$ so that $\pi(x) = x$ AND m is 1-1 on S.*

Proof Initially, let m be the all-0 word. We shall select bits $i_1, \ldots, i_r$ to be set to one, one by one. Let m_j be the appearance of m after j bits have been set and let $\pi_j(x) = x$ AND m_j. Let $P_j = \{\{x,y\} \in S | \pi_j(x) = \pi_j(y)\}$. Suppose $i_1, i_2, \ldots, i_j$ have been selected. If we pick i_{j+1} at random, for any particular pair in P_j, the probability that the pair is in P_{j+1} is at most $1-\delta$. Thus, by an averaging argument, we can pick i_{j+1}, so that $|P_{j+1}| \leq (1-\delta)|P_j|$. Since $|P_0| = \binom{n}{2}$, we can ensure that $|P_r| \leq (1-\delta)^r n^2$, so $r = O(\log n)$ suffices for π_r to be 1-1 on S, if we, for each j, pick i_{j+1} so that $|P_{j+1}|$ becomes as small as possible. We should now argue that we can do this within the stated time bound. For this, we maintain the equivalence classes induced by π_j in a family of linked lists. For any particular choice of i_{j+1}, the finer equivalence classes induced by π_{j+1} can be found in linear time by a linear pass through these lists; this is because each equivalence class is split in (at most) two by the additional bit added. Let the new equivalence classes be $E_1, E_2, \ldots, E_s$. Then, if i_{j+1} is the chosen bit positions, we will have $|P_{j+1}| = \sum_{i=1}^{s} |E_i|(|E_i| - 1)$; this value is easily determined in linear time. Thus, for any particular choice of i_{j+1}, we can find the corresponding value of $|P_{j+1}|$ in time $O(n)$, so we can find the best choice in time $O(wn)$. Since we only have to choose $O(\log n)$ bit positions, the time bound stated in the lemma follows. $\Box$

4 A static dictionary with fast deterministic initialization

We now present our scheme for static dictionaries, proving Theorem 2 stated in the introduction. As lemmas, we need two of the previous results on initializing static dictionaries deterministically. The first is from [20]:

Theorem 9 (Raman) *A static dictionary using linear space and with worst case constant query time for a set of keys $S \subseteq \{0,1\}^w$ of size n can be constructed in worst case deterministic time $O(n^2 w)$.*

Although Andersson does not state the following lemma explicitly in [1], it is the essence of the proof of his Observation 1.

Lemma 10 (Andersson) *Suppose a linear space, constant query time static dictionary for a set of keys S can be constructed in time $f(n)w^{O(1)}$ for some* *function $f(n) \geq n$. Then for any given $\epsilon > 0$, we can also construct such a dictionary in time $f(n)n^{\epsilon}$.*

Our scheme for storing a set of keys $S \subseteq \{0,1\}^w$ is the following. First, using Lemma 6, we apply a good error correcting code to all keys in S. Let the set of the encoded keys be S'. The keys in S' are strings of length $w' = O(w)$, so we can operate on them with unit cost. We apply Lemma 8 and construct the mask $m \in \{0,1\}^{w'}$, where only bits $i_1 < i_2 < \cdots < i_r$ are set. Apply the function $x \to x$ AND m to all the keys in S' and let the resulting set of keys be S''.

Let $k_j = i_{j\epsilon \log n}$, for $j \in \{1, \ldots, s\}$, with $s = \frac{r}{\epsilon \log n}$ (we assume for convenience that $\epsilon \log n$ is an integer that divides r). Since $r = O(\log n)$, we have $s = O(1)$.

Now view each key $x = x[1]x[2] \ldots x[w']$ in S'' as the following string of length s:

$$x[1 \ldots k_1]\ x[k_1 + 1 \ldots k_2] \ldots x[k_{s-2} + 1 \ldots k_{s-1}]\ x[k_{s-1} + 1 \ldots k_s].$$

Thus, the symbols of the string are substrings of x of varying length. The individual symbols of the string can be extracted in constant time using the standard instruction set. We will store our keys in a *path compressed trie* representing these strings. Each leaf of the trie corresponds to a transformed key; we put in each leaf a pointer to the corresponding original key of S. Each node of the trie contains a subset of the possible symbols at some position, and for each symbol present, an associated pointer to a son of the node. We represent the node by a static dictionary implemented using Theorem 9. Thus, the entire data structure will clearly use space $O(n)$ (we do not mention this in the statement of the theorem, but note that the constant in the big-O is even independent on ϵ). By Theorem 9, we can initialize a trie node of size v in time $O(v^2 w')$. However, by construction, only $\epsilon \log n$ bit positions can have a non-zero value for any particular symbol position, so each trie node has size less than n^{ϵ}. Thus, a node of size v can be initialized in time $O(vn^{\epsilon}w')$ and since the sizes of the nodes sum to n, the entire trie can be initialized in time $O(n \cdot n^{\epsilon} w') = O(n^{1+\epsilon} w')$.

We now have a construction with deterministic initialization time $O(n + w'n \log n + n^{1+\epsilon} w') = O(n^{1+\epsilon} w)$. It is easily checked that we can answer queries in constant time; given an input x, we apply the same sequence of transformations to it as we did to the keys of S; look up the transformed key in the (constant depth) trie, and, if we get to a leaf, check if the value in the leaf matches x.

We now finalize the construction by using Lemma 10 and get a solution with deterministic initialization time $O(n^{1+2\epsilon})$. Since ϵ was arbitrary, we are done.

5 Dynamization

In this section we sketch how to dynamize our solution so that we can handle insertions and deletions deterministically in worst case time $O(n^{\epsilon})$ and lookups deterministically in worst case time $O(1)$, proving Theorem 1 stated in the introduction. Apart from some tuning of parameters, the dynamic solution is a standard dynamization of the static solution (see Mehlhorn and Tsakalidis [19, Section 10] for a survey on dynamization). The concrete details of the dynamization are copied more or less verbatim from Thorup's elegant description of his merge heaps [25].

We first construct a solution with a fixed capacity N, space $O(N)$ and with update time $O(N^{\epsilon})$. Given a parameter $\epsilon = 1/k$, we keep in the dynamic structure k substructures D_1, D_2, D_3, $\ldots$, D_k. The structure D_i consists of 2 static dictionaries D_i^1 and D_i^2, one or both may be empty. If D_i^1 is not empty, it has size between $N^{(i-1)\epsilon}$ and $N^{i\epsilon}$. If D_i^2 is not empty, it has size between $N^{(i-1)\epsilon}$ and $N^{(i-1)\epsilon} + 2N^{(i-2)\epsilon}$

If D_i^1 and D_i^2 are both non-empty, we are somewhere in the process of constructing a dictionary of their union, using our efficient deterministic initialization scheme.

When doing a lookup, we simply look in all dictionaries in the structure. There are a constant number, so the worst case complexity is constant.

When doing an insert of a key x, we put x in the empty one-word slot D_1^2, rebuild D_1^1 as the union of D_1^1 and D_1^2, and throw away D_1^2. If the size of D_1^1 grows to be more than N^{ϵ}, we move D_1^1 to D_2^2, which will be empty. In general, for each insert, we do a $1/N^{(i-1)\epsilon}$ fraction of the construction of the union dictionary at level i. If this completes the construction, we throw away D_i^1 and D_i^2 and put in the union in the place of D_i^1, unless it is too large, in which case we move it to D_{i+1}^2. The complexity of the operation is $O(N^{2\epsilon})$. Since ϵ was arbitrary, this is as desired.

When doing a delete, we locate the element and mark it as deleted in its dictionary and in the partially constructed union. When we start a new construction of a union, we don't include any elements which were marked at the start. The complexity of the delete operation is $O(1)$.

Getting a solution with space bound $O(n)$, time bound $O(n^\epsilon)$ for insertions and deletions and time bound $O(1)$ for lookups from a solution with space bound $O(N)$, time bound $O(N^\epsilon)$ for insertions, and time bound $O(1)$ for deletions and lookups, is standard global rebuilding.

6 A solution with no trace of universal hashing

In the introduction, we presented the error correction/clustering-combination as an alternative to universal hashing. Note, however, that our solution uses Raman's static structure as a substructure and since this is based on universal hashing, this presentation may seem misleading. However, in this section we present a different implementation which does not use Raman's structure and where universal hashing plays no role whatsoever. The solution has the flaw of not using a standard instruction set, so it is best understood in the cell probe model of computation. It is mainly interesting from a philosophical point of view, as the first dictionary with linear space and constant query time (in the cell probe model) for general values of n and w with no trace of universal hashing anywhere.

The non-standard unit cost instructions we need are $\text{ERROR} : \{0,1\}^w \rightarrow \{0,1\}^{O(w)}$ which computes an explicit good error correcting code, such as a Justesen code (see, for instance, MacWilliams and Sloan [18]), and COLLECT, which takes as inputs a word containing a mask m and a word x and returns the concatenation of the bits of x which are marked by m. For instance, $\text{COLLECT}(0100100001001001, 1101011110111010) = 10010$. The ERROR instruction is used to replace multiplicative error correcting codes, so that we will not be accused of sneaking in universal hashing through the back door. The COLLECT instruction plays a more indispensable role.

In analogy with the previous scheme, we start by applying ERROR to each key of the set S and get the set S' of keys of length $w' = O(w)$. We apply Lemma 8 to S' and construct the mask m. Now we use the instruction $x'' = \text{COLLECT}(m, x')$ on all keys x of S' and get the set S''. The keys in S''

have length $w'' = O(\log n)$. Now, we store the keys of S'' using Tarjan and Yao's *double displacement* scheme [24]. This is a scheme, with no trace of universal hashing (it was invented in 1978), for the static dictionary problem for a set $S \subseteq \{0,1\}^w$ of size n with space $O(n)$ and query time $O(w/\log n)$. In our case, $S'' \subseteq \{0,1\}^{w''}$ with $w'' = O(\log n)$, so the query time becomes $O(1)$. This completes the construction.

As stated in their paper, Tarjan and Yao's has deterministic initialization time $O(n^2)$. Thus, it seems that this alternative solution not only uses a non-standard instruction set, but also kills the main point of this paper, efficient deterministic operations. However, it is actually possible to improve Tarjan and Yao's scheme so that the initialization time becomes $O(n^{1+\epsilon})$. The key point to observe is that the high time bound of Tarjan and Yao's algorithm is due to heavy string matching which is done using techniques which were state of the art in 1978. Using more modern data structures for maintaining strings, in particular Sahinalp and Vishkin's dynamic pattern matching algorithm [23] yields the improved initialization time. Of course, the main point of this section is to present a static dictionary with $O(n)$ space and $O(1)$ query time with no trace of universal hashing, and this main point is safe, no matter how much time is used for initialization, so we will not present the somewhat technical details of the improved initialization here.

7 Improved perfect hashing circuits

We show the following theorem, from which Theorem 4 stated in the introduction follows (for the first bound, put $l = \log(1/3\epsilon)$; the theorem is valid, even for $m > 2k$, for the rest, put $l = 1 + \log k$).

Theorem 11 *For every $w, k \leq w$, $l \geq 1$, and $m \geq k$, and for every $S \subseteq \{0,1\}^w$ of size 2^k, there is a Boolean circuit C of size $O(w + k2^{2k-m+m/2^l} + lm)$ mapping $\{0,1\}^w$ to $\{0,1\}^m$ so that the function defined by C is 1-1 on S.*

Proof

Gelfand, Dobrushin, and Pinsker [12] have shown that good error correcting codes with linear sized encoding circuits exists, that is, for any w, there is a circuit E of size $O(w)$ mapping $\{0,1\}^w$ to $\{0,1\}^{w'}$ with $w' = O(w)$ so that for all $x, y \in \{0,1\}^w$, the Hamming distance between $E(x)$ and $E(y)$ is at least $\delta w'$ for a certain constant $\delta > 0$. A more recent reference for such codes is Spielman [21]; his codes can also be decoded in linear time, but we only need the encoding circuits.

Step 1. As basis for our perfect hashing circuit we take a copy of E, applied to the input vector. Let $z_1, z_2, \ldots, z_{w'}$ be the output bits of E.

As in Lemma 8, we will select a subset $z_{i_1}, z_{i_2}, \ldots, z_{i_r}$ of the outputs with $r = \lceil 2k/\log \frac{1}{1-\delta} + 1 \rceil$, and disregard the rest. Call the resulting circuit E', mapping $\{0,1\}^w \to \{0,1\}^r$. We shall make sure that E' is 1-1 on S. Let E_j be the circuit mapping the input to $z_{i_1}, z_{i_2}, \ldots z_{i_j}$ Let $P_j = \{\{x,y\} \in S | E_j(x) = E_j(y)\}$. Suppose $z_{i_1}, z_{i_2}, \ldots, z_{i_j}$ have been selected. If we pick $z_{i_{j+1}}$ at random, for any particular pair in P_j, the probability that the pair is in P_{j+1} is at most $1-\delta$. Thus, by an averaging argument, we can pick $z_{i_{j+1}}$, so that $|P_{j+1}| \leq (1-\delta)|P_j|$. Since $|P_0| = \binom{2^k}{2}$, we can ensure that $|P_r| \leq (1-\delta)^r 2^{2k} < 1$, and since it is an integer, it is 0, so $E' = E_r$ is 1-1. So far, we have constructed a circuit of size $O(w)$ mapping S 1-1 to $r = O(k)$ bits.

Step 2. We now construct another circuit with r inputs and m outputs which will be 1-1 on $E'(S)$.

Again we use an error correcting circuit, defining an error correcting code with relative minimum distance at least δ, this time with number of inputs r, number of outputs $r' = O(r)$, and size $O(r) = O(k)$. Let the outputs be $q_1, \ldots, q_{r'}$. We define gates $o_1, o_2, \ldots, o_{m-1}$ as XOR-gates of odd fan-in $l' \geq \lceil (l+1)/\log(\frac{1}{1-2\delta}) \rceil$, each adding size l' to the circuit, each taking a subset of the q_i's as inputs (here, we are essentially using one of the more powerful clustering functions from [2]). We will fix the inputs of the o_i's iteratively. Suppose $o_1, \ldots, o_i$ have been fixed and let C_i be the circuit mapping the input to $o_1, o_2, \ldots o_i$. Let $P_i = \{\{x,y\} \in E'(S) | C_i(x) = C_i(y)\}$. Now suppose we pick the l' inputs of o_{i+1} randomly from the q_j's. As observed in [2], for each $\{x,y\} \in P_i$, the probability that $\{x,y\}$ is in P_{i+1} is at most $\frac{1}{2}(1+(1-2\delta)^{l'}) \leq \frac{1}{2}(1+2^{-(l+1)})$. Thus, we can pick a setting of the inputs so that $|P_{i+1}| \leq \frac{1}{2}(1+2^{-(l+1)})|P_i|$, and thus ensure that $|P_r| \leq 2^{2k-m+1}(1+2^{-(l+1)})^{m-1} \leq 2^{2k-m+1+m/2^l}$. Call this last quantity u. Let C_1 be the circuit, resulting from disregarding all outputs but the o_i's. We can remove a subset T of size u from $E'(S)$ so that C_1 is 1-1 on $E'(S) - T$. Let C_2 be the obvious perfect hashing circuit for T of size $O(ur)$. There is also an obvious circuit of size $O(ur)$ deciding membership of T. Now let $C(x) = 0 \circ C_2(x)$ if $x \in T$ and $C(x) = 1 \circ C_1(x)$ otherwise.

Step 3. The desired circuit is E' composed with C. The size of this circuit is $O(w+k+ml+ku) = O(w+k2^{2k-m+m/2^l}+lm)$, as desired. $\Box$

8 Acknowledgement

I would like to thank Martin Dietzfelbinger for very helpful discussions about the material of Section 2.

References

[1] A. Andersson. Faster deterministic sorting and searching in linear space In *37th IEEE Symposium on Foundations of Computer Science*, pages 135–141, Burlington, Vermont, 1996.

[2] A. Andersson, P.B. Miltersen, S. Riis, and M. Thorup. Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{\log n/\log\log n})$ is necessary and sufficient. In *36th IEEE Symposium on Foundations of Computer Science*, pages 538–546, Burlington, Vermont, 1996.

[3] A.M. Ben-Amram and Z. Galil. When can we sort in $o(n \log n)$ time? In *34th IEEE Symposium on Foundations of Computer Science*, pages 538–546, Palo Alto, California, 1993.

[4] J. Bierbrauer, I. Johansson, G. Kabatianskii, and B. Smeets. On families of hash functions via geometric codes and concatenation. In: *Advances in Cryptology - CRYPTO '93, Lecture Notes in Computer Science* vol. 773, pp. 331–342, Springer, 1993.

[5] A. Brodnik and J.I. Munro. Membership in constant time and minimum space. In: *Proceedings of the 1st European Symposium on Algorithms, Lecture Notes in Computer Science*, Vol. 855, page 72, 1994.

[6] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[7] M. Dietzfelbinger and F. Meyer auf der Heide, Dynamic hashing in real time, in: J. Buchmann, H. Ganzinger, W. J. Paul (Eds.): *Informatik ·*

Festschrift zum 60. Geburtstag von Günter Hotz, Teubner-Texte zur Informatik, Band 1, B. G. Teubner, 1992, pp. 95–119. (A preliminary version appeared under the title "A New Universal Class of Hash Functions and Dynamic Hashing in Real Time" in *ICALP'90.*)

[8] M. Dietzfelbinger, J. Gil, Y. Matias and N. Pippenger. Polynomial hash functions are reliable. In: *Proc. 19th Int'l. Colloq. on Automata, Languages and Programming, Lecture Notes in Computer Science* Vol. 623, pages 235-246, Springer-Verlag, July 1992.

[9] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. Technical Report 513, Fachbereich Informatik, Universität Dortmund, Dortmund, Germany, 1993.

[10] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer Auf Der Heide, H. Rohnert, R .E. Tarjan, Dynamic perfect hashing: upper and lower bounds, *SIAM J. Comput.* **23** (1994) 738–761.

[11] M. Dietzfelbinger. Universal Hashing and k-wise Independent Random Variables via Integer Arithmetic without Primes. In *Proc. 13th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science* vol. 1046, pp. 569-580, Springer, 1996.

[12] S.I. Gelfand, R.L. Dobrushin, and M.S. Pinsker. On the complexity of coding. In *Second International Symposium on Information Theory*, pages 177–184, Akademiai Kiado, Budapest, 1973.

[13] A. Fiat, M. Naor, J.P. Schmidt and A. Siegel, Non-Oblivious Hashing. In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 367–376, 1988.

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

[15] M.L. Fredman and D.E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436, 1993.

[16] M.L. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533–551, June 1994.

[17] O. Goldreich and A. Wigderson, On the Circuit Complexity of Perfect Hashing, *Electronic Colloquium on Computational Complexity*, TR96-04, 1996.

[18] F.J. MacWilliams and N.J.A Sloane. *The Theory of Error-Correcting Codes*. North Holland, Amsterdam, 1977.

[19] K. Mehlhorn and A. Tsakalidis. Data Structures. In: *Handbook of Theoretical Computer Science, Vol. A: Algorithms and complexity*, pp. 303–341, MIT Press, 1994.

[20] R. Raman. Priority queues: Small, monotone, and trans-dichotomus. In *Proceedings 4th European Symposium on Algorithms*, volume 1136 of *Lecture Notes in Computer Science*, pages 121–137. Springer-Verlag, 1996.

[21] D.A. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings 27th annual ACM symposium on the theory of computing*, pages 388–397, Las Vegas, Nevada, 1994.

[22] D.R. Stinton. On the connections between universal hashing, combinatorial designs and error-correcting codes. *Electronic Colloquium on Computational Complexity*, TR95-052, 1995.

[23] S.C. Sahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labelling paradigm. In *36th IEEE Symposium on Foundations of Computer Science*, pages 320–328, Burlington, Vermont, 1996.

[24] R.E. Tarjan and A.C. Yao. Storing a sparse table. *Communications of the ACM*, 22(11):606–611.

[25] M. Thorup. On RAM priority queues. In *7th ACM-SIAM Symposium on Discrete Algorithms*, pages 59–67, Atlanta, Georgia, 1996.

Recent BRICS Report Series Publications

RS-97-17 **Peter Bro Miltersen.** ***Error Correcting Codes, Perfect Hashing Circuits, and Deterministic Dynamic Dictionaries*****. June 1997. 19 pp. To appear in** ***The Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*****, SODA '98.**

RS-97-16 **Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos.** ***Linear Hashing*****. June 1997. 22 pp. A preliminary version appeared with the title** ***Is Linear Hashing Good?*** **in** ***The Twenty-ninth Annual ACM Symposium on Theory of Computing*****, STOC '97, pages 465–474.**

RS-97-15 **Pierre-Louis Curien, Gordon Plotkin, and Glynn Winskel.** ***Bistructures, Bidomains and Linear Logic*****. June 1997. 41 pp.**

RS-97-14 **Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup.** ***Dictionaries on AC^0 RAMs: Query Time $\Theta(\sqrt{\log n / \log\log n})$ is Necessary and Sufficient*****. June 1997. 18 pp. Appears in** ***37th Annual Symposium on Foundations of Computer Science, FOCS '96 Proceedings*****, pages 441–450.**

RS-97-13 **Jørgen H. Andersen and Kim G. Larsen.** ***Compositional Safety Logics*****. June 1997. 16 pp.**

RS-97-12 **Andrej Brodnik, Peter Bro Miltersen, and J. Ian Munro.** ***Trans-Dichotomous Algorithms without Multiplication — some Upper and Lower Bounds*****. May 1997. 19 pp. Appears in Dehne, Rau-Chaulin, Sack and Tamassio, editors,** ***Algorithms and Data Structures: 5th International Workshop*****, WADS '97 Proceedings, LNCS 1272, 1997, pages 426–439.**

RS-97-11 **Kārlis Čerāns, Jens Chr. Godskesen, and Kim G. Larsen.** ***Timed Modal Specification — Theory and Tools*****. April 1997. 32 pp.**

RS-97-10 **Thomas Troels Hildebrandt and Vladimiro Sassone.** ***Transition Systems with Independence and Multi-Arcs*****. April 1997. 20 pp. Appears in Peled, Pratt and Holzmann, editors,** ***DIMACS Workshop on Partial Order Methods in Verification*****, POMIV '96, pages 273–288.**