Slightly more practical deterministic static dictionaries

Manuel Stoeckl*

Abstract

The current best construction algorithms for perfect hash functions and dictionaries are efficient but rely on randomization, with a few exceptions. There do exist deterministic constructions for perfect hash functions (and by extension, static dictionaries), with near-linear construction time, but these are not competitive for low latency access, requiring more than the minimum one input-dependent read operation for the worst case in the cell probe model to evaluate a perfect hash function.

For integers u and n, we describe a deterministic construction of a simple perfect hash function from $[2^u]$ to $[O(n)]$ for any set S of n keys in $[2^u]$, whose evaluation requires exactly one dependent load of an $O(\log n)$-bit region, and whose total space usage is $O(n \log n)$ bits. Construction takes $O(nu \log n)$ time. Our key contribution in this is a faster deterministic selection of standard ”odd-multiply-shift” hash function mapping a set S in $[2^u]$ of size n to $[O(n^2)]$ with no more collisions than expected; similar results with a smaller range are possible using the ”odd-multiply-add-shift” and ”odd-carryless-multiply-shift” families.

1 Introduction

A static dictionary is a data structure constructed from a set P of n distinct key-value pairs in a universe $U \times V$, supporting item retrieval queries:[1]

Query(x) For some key $x \in U$, if $(x, v) \in P$, report the value v, otherwise $\perp$.

A common (but by no means the only) way to construct a static dictionary with $O(1)$-worst-case access time in the word-RAM or cell probe models is to construct a perfect hash function f from U to $\{0, \dots, t-1\}$, for some $t = O(n)$, which is injective on $S := \{x : (x, v) \in P\}$, and use it to index into an array containing, for each $x \in S$, the value (x, v) at index $f(x)$.[2]

Simple and randomized $O(n)$-time constructions for $O(1)$-query time perfect hash functions and static dictionaries have been known since [FKS84], but the development of near-linear time deterministic perfect hash and dictionary constructions has taken longer, culminating in [Ruz08a]'s algorithm that uses $O(\text{SORT}(n) \log \log n)$ time in certain models, where $\text{SORT}(n)$ is the time needed to sort an array of n entries. However, these dictionaries vary in how efficient they are to query, and usually require reading multiply (but still $O(1)$) values from memory.

The best possible worst-case number of dependent memory read operations for a query – distinct words or cache-lines that are read, excluding $O(1)$ words that are read on every query – for a static dictionary using $O(n \operatorname{polylog}(n, |U|))$-space dictionary is two, and for a perfect hash function using similar space is one. [**Standard space lower bound proves this for hash functions; need to cite the appropriate dictionary** !!!

*No affiliation, undeclared location. Contact: research@mstoeckl.com

[1]Another common definition is to have a static dictionary support *membership* queries, which just report whether a key is inside the given set.

[2]Note that constructing a perfect hash function from a dictionary is also easy to do.

cell probe lower bound.]

This paper describes a deterministic algorithm to identify an odd-multiply-shift (OMS) hash function whose collision count (Def: 1.5) is not more than expected for a random OMS hash function, and uses that to describe a *perfect* hash function for a given set S of n elements from $U = \{0,1\}^u$ to $\{0,\dots,n-1\}$ which can be constructed in $O(nu\log n)$ time, and requires making only *one* dependent probe to memory. Using this one can construct an efficiently evaluable dictionary with key universe $U = \{0,1\}^u$.

1.1 Definitions and main results

We identify $\{0,1\}^u$ with $\{0,\dots,2^u-1\}$, define $x \operatorname{div} y = \lfloor x/y \rfloor$, and $\odot$ to be the carryless-multiplication/$\mathbb{F}_2$-polynomial-multiplication operator.

Definition 1.1. A hash family $\mathcal{H}$ of functions from $\{0,1\}^u$ to $\{0,1\}^s$ is *α-approximately universal* if for all distinct $x,y \in \{0,1\}^u$, for h chosen uniformly at random from $\mathcal{H}$:

$$\Pr[h(x) = h(y)] \leq \alpha/2^s\,.$$

A 1-approximately universal hash family is *universal*. (See also: [CW79].)

Definition 1.2. The Odd-Multiply-Shift (OMS) hash family $\mathcal{H}^{\mathrm{OMS}}$ from $\{0,1\}^u$ to $\{0,1\}^s$ contains all functions:

$$h_a : x \mapsto (ax \bmod 2^u \operatorname{div} 2^{u-s}) \qquad \text{for odd } a \in \{1,3,5,\dots,2^u-1\}$$

This family is 2-approximately universal (see [DHKP97, Lemma 2.4] and Lemma 2.3).

Definition 1.3. The Odd-Multiply-Add-Shift (OMA) hash family $\mathcal{H}^{\mathrm{OMA}}$ from $\{0,1\}^u$ to $\{0,1\}^s$ contains all functions:

$$h_{a,b} : x \mapsto ((ax+b) \bmod 2^u \operatorname{div} 2^{u-s}) \qquad \text{for odd } a \in \{1,3,\dots,2^u-1\} \text{ and all } b \in \{0,\dots,2^{u-s}-1\}$$

This family is universal (Lemma 3.2); other variants of it have different properties[Tho15, Section 3.3].

Definition 1.4. The Odd-Carryless-Multiply-Shift (OCM) hash family $\mathcal{H}^{\mathrm{OCM}}$ from $\{0,1\}^u$ to $\{0,1\}^s$ contains all functions:

$$h_{a,b} : x \mapsto (a \odot x \bmod 2^u \operatorname{div} 2^{u-s}) \qquad \text{for odd } a \in \{1,3,\dots,2^u-1\}$$

This family is universal (Corollary 4.3); it can be considered a variant of Toeplitz hashing[Kra94].

Definition 1.5. For a set $S \in \{0,1\}^u$ and function $f : \{0,1\}^u \to \{0,1\}^s$, the *collision count* $\kappa(f,S)$ of f is the number of unordered pairs of distinct $x,y \in S$ for which $f(x) = f(y)$. Calculated differently, it is $\kappa(f,S) := \sum_{z \in \{0,1\}^s} \binom{|f^{-1}(z) \cap S|}{2}$.

Theorem 1.6. *Given integers n,u,s, for any subset $S \subseteq \{0,1\}^u$ of size n, there is an algorithm taking $O(nu\log n)$ time (in Word-RAM, Definition 1.9) and $O(nu)$ additional space to identify an multiply-shift hash function $f : \{0,1\}^u \to \{0,1\}^s$ with collision count $\kappa(f,S) \leq \frac{2}{2^u}\binom{n}{2}$.*

Theorem 1.7. *A variant of the algorithm of Theorem 1.6 can, with the same time and space, select a multiply-shift-add function f with collision count $\kappa(f,S) \leq \frac{1}{2^u}\binom{n}{2}$.*

Theorem 1.8. *A variant of the algorithm of Theorem 1.6 can, with the same time and space, select a carryless-multiply-shift function f with collision count $\kappa(f,S) \leq \frac{1}{2^u}\binom{n}{2}$.*

Theorem 8.3. *For n, u positive integers with $u \geq \log n$, implementing Algorithm 8 using either OMA or OCM hashes will construct a minimal perfect hash function f for a n-element subset S in $\{0,1\}^u$ to $[n]$, using $O(nu \log n)$ time, $O(nu)$ auxiliary space; the hash function f itself uses $\leq 5.66n\lceil \log n \rceil + O(u + \log n)$ bits of space and can be evaluated with exactly one dependent read of a $\lceil \log n \rceil$-bit word; there are $O(u + \log n)$ bits of parameters that are unconditionally used). OMS hashes can also be used but double the bound on the space usage.*

1.2 Notation

In this paper, we interpret $\log(x)$ as the base-2 logarithm of x, $\ln(x)$ as the base-e logarithm, and $[k] := \{0, 1, \ldots, k-1\}$. We sometimes interpret bit strings $\{0,1\}^u$ as base-2 integers in $[2^u)$. $\mathbb{1}\{\texttt{event}\}$ is the function which has value 1 iff `event` holds, and 0 otherwise. For two strings of bits p, q, their concatenation is $p.q$; for example, $(1,2,3).(7) = (1,2,3,7)$.

Bit strings interpreted as *little*-endian; $(0,1,1,1) = 14$; the ith (counting from zero) entry of the bit string is the ith (counting from zero) least significant bit of the base two representation of the value.

The function: BREV reverses the bits in an integer

The function: TZCNT : $\{0,1\}^u \to \{0, \ldots, u\}$ counts the number of trailing/least-significant zeros of the input

The binary operator $a \operatorname{sel} [b, c)$ outputs $a \bmod 2^c \operatorname{div} 2^b$, and thus selects least significant bits b through $c-1$ of the binary representation of a. Note that $a \operatorname{sel} [b, c)$ also equals $a \operatorname{div} 2^b \bmod 2^{c-b}$.

In this paper, we consider two models of computation. On actual computers, the input and output of a hash function are often small (≤ 256 bits) and the speed of operations on word-size quantities may vary significantly, so even when $n = 2^{30}$, factor of $\log n$ differences in asymptotic bounds may be outweighed by hardware differences.

Definition 1.9. Unit cost word-RAM (with constant-time multiplication and carryless multiplication); this is similar to the executed instruction count of a program.

Definition 1.10. Finite-tape Turing machine: this imposes a high penalty on random access to memory and measures costs with bit level precision.

1.3 Applicability

Randomized dictionary and perfect hash function construction algorithms can have much better performance, in construction time, size, and query time, than the current best known deterministic constructions. In general, the results of this paper are not competitive for practical use, but there are a few exceptions where determinism may be worth the slower construction. For example, if the system being used does not have any reliable and private source of randomness, or any strong cryptographic primitives, and the input set of keys is chosen adversarially. Deterministic perfect hash function constructions are also necessarily reproducible and may be useful in file formats which need fast random access and encoders that with reproducible output and strong worst-case construction time guarantees. The hash function selection algorithms in this paper use the method of conditional expectations and can always find a function with no more than the expected number of collisions, even if only one function in the family has this property; random sampling can only reliably find a function with a property if a large enough fraction of functions have the property. This may lead to very slight space improvements for designs that already use one of the OMS, OMA, or OCM families.

1.4 Related work

[FKS84] (FKS) introduced a simple and general randomized construction for a static dictionary (equivalently, for a perfect hash function) on a set S of n keys, using $O(n)$ expected time (in a unit-cost-word-RAM

model) to construct, $O(n)$ words of space, and supporting worst-case $O(1)$-time queries (assuming small enough keys that can be hashed in $O(1)$ time). They also observed that when using specific families of hash functions, their dictionary can be deterministically constructed in $O(n^3 \log u)$ time, for u-bit keys. This is close to optimal: [FK84, Theorem 2] proved that when $u \geq (2+\varepsilon)\log n$, any perfect hash function from $\{0,1\}^u$ to $[O(n)]$ requires $\geq n \log e + \log\log u - O(\log n)$ bits of space. (On the other hand, $O(1)$-access time dictionaries can be built which do not require *any* additional space over that needed to store the uncompressed key/value pairs[FN93]; by compressing keys, one can obtain down to [**cite**] space.) Later work !!!
[**cite**] has gotten close to these bounds. !!!

While most research on dictionaries over the last fifty years has focused on randomized, dynamic dictionaries or randomized perfect hash function constructions, a few papers have found results for static deterministic dictionaries (equivalently, perfect hash functions) which get closer to the randomized performance of [FKS84]. [**cite: no competitive dynamic deterministic dictionaries**] Among these are: [Ram95], !!!
which found that with the odd-multiply-shift hash family, [FKS84]'s construction time can be improved to $O(n^2 u)$; [AN96], giving an $O(nu(\log n)^4)$ time construction; [HMP01], which assuming one can construct and efficiently evaluate an explicit linear code with high rate and relative distance ([**doable with mediocre parameters**]), gives an $O(n \log n)$ time construction. !!! Slightly later, a series of work ending in [Ruz08b, Ruz08a, Ruz09] identified improved dictionary constructions which run in $O(??)$ time.

There are a few main directions to randomized constructions of perfect hash functions with $O(1)$ worst-case or expected query time. The FKS scheme uses a secondary table of hash functions; this can be compressed ([**cite: something via AlonN96**]) to reduce the number of bits needed to $O(??)$. [**cite**] has !!!
hash-displace-and-compress. [**cite**] observes splitting-and-filtering-approach – if majority of keys handled !!!
immediately, remainder can afford more space/time expensive construction.[3] !!! A randomized static dictionary design which could promise even lower access latency (two parallel reads, instead of sequential) is to use two hash functions h_1, h_2 to query locations in an array, and combine the values $A[h_1(k)]$ and $A[h_2(k)]$ in the array to produce the value associated with k [CHM92]. This is similar to Cuckoo hashing, and there have been a variety of improvements to the scheme since then[**cite the notable improvements**]; this is sim- !!!
ilar to Cuckoo hashing. Unfortunately, the hash families required for this are not well as well understood as the plain universality criterion required for the FKS scheme: [**cite**], and no near-linear derandomized !!!
construction is known. [**Any $O(n)$ bit constructions?**] !!!

[**Note: the publicly available "Constructing Efficient Dictionaries in Close to Sorting Time" paper is dated from July 2009 and may not match R09b, so its citation may need adjustment.**] !!!

[**Find Turing machine time bounds for carryless multiplication / $\mathbb{F}_2$ polynomial multiplication, since this influences asymptotic OCM evaluation time**] !!!

A few open problems related to this work are listed in Section 9.

[3] Splitting the dictionary into large "easy" and small "hard" regions, in some way, is a common theme in both old and recent theoretical work on dynamic dictionary constructions (see for example [BCF+21]), second perhaps only to the use of queues to reduce risk and spread work between operations.

Source	Output	Construction time (word-RAM)	Evaluation time (Turing)	Space
Mod-by-prime [FKS84, Lemma 2]	$O(n^2 \log u)$	$O(n^3 \log u)$,	$O(u \log\log n)$	$O(\log n + \log u)$
Odd-multiply-shift[Ram95]	$O(n^2)$	$O(n^2 u)$	$O(u \log u)$	$O(u)$
OMA or OMS, this paper	$O(n^2)$	$O(nu \log n)$	$O(u \log u)$	$O(u)$
OCM, this paper	$O(n^2)$	$O(nu \log n)$	$O(u(\log\log n)^{1+o(1)})$	$O(u)$
Linear code bit extraction[HMP01]	$n^{O(1)}$	$O(n \log n)$,	$O(u \log u)$?	$O(\log n + \log u)$
Multiplicative [Ruz08b, Section 3.1]	$O(n^2)$	$O(n^2 \log n)$	$O(u \log u)$	$O(u)$??
Multiplicative [Ruz08b, Section 3.2]	$O(n^4)$	$O(nu(\log n)^2)$	$O(u \log u)$	$O(u + \log n)$
Shift-plus-multiply, [Ruz09], exact inversion count	$O(2^{u/2+\log n})$	$O(??)$	$O(u \log\log n)$	$O(\log n)$
Shift-plus-multiply, [Ruz09], approx. inv. count	$O(2^{u/2+\frac{\log n}{\log \frac{3}{2}}})$	$O(??)$	$O(u \log\log n)$	$O(\log n)$
[AN96]	???			

Table 1: Deterministic construction of hash functions for universe reduction on input universe $\{0,1\}^u$; with word size $w \geq u$. This table shows primitives, which may either stand alone or be combined or iterated to obtain better bounds. TODO: double check and document all of these

Row	Source	Construction time (word-RAM)	Memory usage (bits)	Maximum indirect read count
1	**OMS/OMA/OCM and single displacement, this paper**	$O(nu \log n)$	$O(n \log n + u)$	1
2	Linear code bit extraction and trie via double displacement[HMP01]	$O(n \log n)$ †	$O(n \log n + \log n \log u)$	≥ 2 and $O(1)$
3	Shift-plus-multiply with iterated relaxed displacement[Ruz08a, Ruz09]	$O(n???)$		≥ 3 and $O(1)$
4	Tri-level [AN96]	$O(nu(\log n)^4)$	$O(n + \log n \log u)$	2
5	Bi-level [FKS84] deterministic	$O(n^3 u)$		1

Table 2: Deterministic perfect hash function constructions (from $[2^u]$ to $[O(n)]$), with construction time calculated assuming word size $w \geq u$. †: requires pre-calculation of a linear code. TODO: double check and document all

2 Properties of the odd-multiply-shift (OMS) hash family

Observation 2.1 (On multiplication mod 2^k)**.** *For integers $i \leq k$ and nonzero $x \in [2^k)$, assume $x = z2^i$ for some odd integer z. If a is a uniformly random integer in $[2^k)$ (or just in $[2^{k-i})$), then the distribution of*

$ax \bmod 2^k$ is the uniform distribution over multiples of 2^i in $[2^k)$. If a is a uniformly random odd *integer in $[2^k)$ (or just in $[2^{k-i})$), then $ax \bmod 2^k$ is distributed uniformly randomly over* odd *multiples of 2^i in $[2^k)$.*[4]

Lemma 2.2. *Let $j \le k \le l$ be nonnegative integers. Let I be an interval containing exactly of 2^k elements in $\mathbb{Z}/2^l\mathbb{Z}$. For any integer $x \in \mathbb{Z}/2^l\mathbb{Z}$, the set S of values $(x + 2^j\mathbb{Z})/2^l\mathbb{Z}$ intersects I exactly 2^{k-j} times.*

Proof. Split I into 2^{k-j} consecutive disjoint intervals $I_1, \ldots, I_{2^{k-j}}$ each of length 2^j. Each interval I_i for some i contains one element for each equivalence class $\bmod 2^j$ and so intersects S exactly once. □

The following result has long since been known[DHKP97, Lemma 2.4], but we reprove it here for completeness. [**TODO: move most of this (and most of the paper) into an appendix to keep the main line focused on the interesting parts**] !!!

Lemma 2.3. *The OMS hash family from $\{0,1\}^u$ to $\{0,1\}^s$ is 2-approximately-universal.*

Proof. Consider any pair x, y of distinct values in $\{0,1\}^u$. Since $x - y \neq 0$, let z, i be integers with z odd so that $x - y \bmod 2^u = z2^i$. Let a be a uniformly random odd integers in $[2^u)$ and let h_a be the corresponding OMS hash function from $\{0,1\}^u$ to $\{0,1\}^s$. Since $x - y \bmod 2^u = z2^i$, we can write $x = x_h 2^i + x_l$ and $y = y_h 2^i + y_l$ where $x_l = y_l \in [2^i)$, $x_h \in [2^{u-i})$, and $y_h = (x_h + z) \bmod 2^{u-i}$. We have two cases:

1. If $i \ge u - s$, then

$$\begin{aligned} h_a(x) &= (ax_h 2^i + ax_l) \bmod 2^u \operatorname{div} 2^{u-s} \\ &= (ax_h 2^i + ax_l) \operatorname{div} 2^{u-s} \bmod 2^s \\ &= (ax_h \bmod 2^{u-i}) 2^{i-u-s} + ((ax_l) \operatorname{div} 2^{u-s} \bmod 2^s) . \end{aligned}$$

   Since a is odd, the map $z \mapsto az \bmod 2^{u-i}$ is a bijection. Since $x_h \neq y_h$, $(ax_h \bmod 2^{u-i}) \neq (ay_h \bmod 2^{u-i})$. On the other hand, $ax_l = ay_l$. Consequently, $h_a(x) \neq h_a(y)$, so

$$\Pr[h_a(x) = h_a(y)] = 0 \le 2/2^s .$$

2. If $i < u - s$, then the value of $\Pr[h_a(x) = h_a(y)]$ may vary depending on the precise values of x and y. We will prove a uniform and slightly looser upper bound. For any given value of a, we have two subcases. If $ay \bmod 2^u \ge ax \bmod 2^u$, then $h_a(x) = h_a(y)$ implies $a(y - x) \bmod 2^u < 2^{u-s}$, since both $ay \bmod 2^u$ and $ax \bmod 2^u$ must fall in the same contiguous interval of length 2^{u-s} for floor-division by 2^{u-s} to produce the same value. Symmetrically, if $ax \bmod 2^u \ge ay \bmod 2^u$, then $h_a(x) = h_a(y)$ implies $a(x - y) \bmod 2^u < 2^{u-s}$.[5] Thus:

$$\Pr[h_a(x) = h_a(y)] \le \Pr[a(x-y) \bmod 2^u \in \{0, \ldots, 2^{u-s} - 1\} \cup \{2^u - 2^{u-s} + 1, \ldots, 2^u - 1\}] . \quad (1)$$

   Since z is odd, and a is chosen uniformly randomly over odd values in $[2^b)$, the distribution of $az2^i$ is uniform over odd multiples of 2^i in $[2^u)$. The set $M_{2^{u-s},2^u} = \{0, \ldots, 2^{u-s} - 1\} \cup \{2^u - 2^{u-s} + 1, \ldots, 2^u - 1\}$ is a subset of a contiguous interval of length 2^{u-s+1} in $\mathbb{Z}/2^u\mathbb{Z}$, so by Lemma 2.2, the set of possible values of $az2^i$ has at most $2^{u-s+1}/2^{i+1}$ intersections with $M_{2^{u-s},2^u}$, out of 2^{u-i+1} possible options. Therefore,

$$\Pr[h_a(x) = h_a(y)] \le \frac{2^{u-s+1}}{2^{i+1} 2^{u-i+1}} = \frac{2}{2^s}$$

Note that the first case only uses the fact that the hash function parameter a is odd. The second case requires a uniform distribution, and would also work (with minor adjustment) if a were chosen uniformly at random over *all* integers in $[2^b)$ instead of just the odd integers. □

[**Can we construct a *tight* input x,y for every u and s?**] !!!

We will later need to efficiently bound the probability that a given pair x,y of distinct elements collides under an OMS hash function h_a, where h_a is chosen from a specific family of conditional distributions. Define

$$M_{2^{u-s},2^u} := \{0,\ldots,2^{u-s}-1\} \cup \{2^u - 2^{u-s}+1,\ldots,2^u-1\};$$

this is equivalent, $\bmod 2^u$, to $\{-2^{u-s}+1,\ldots,2^{u-s}-1\}$. Let

$$\delta_a(x,y) = \begin{cases} 0 & \text{if } x-y \bmod 2^{u-s} = 0 \\ \mathbb{1}\{a(x-y) \bmod 2^u \in M_{2^{u-s},2^u}\} & \text{otherwise} \end{cases} \tag{2}$$

As argued when proving Eq. 1 in the proof of Lemma 2.3, this is an upper bound on $\mathbb{1}\{h_a(x)=h_a(y)\}$. ($\delta_a(x,y)$ is almost the same as the collision bound used by [Ram95, Lemma 4], differing only in the special case for $x-y \bmod 2^{u-s}=0$.)

Lemma 2.4. *For $u \ge s \ge 1$, consider the OMS hash family $\mathcal{H}^{\mathrm{OMS}}$ from $\{0,1\}^u$ to $\{0,1\}^s$. Let $x,y \in \{0,1\}^u$ be distinct. Let z,i be integers so that $x-y \bmod 2^u = z2^i$ and z is odd. For any odd $\alpha \in \{0,1\}^\star$, if h_a is chosen uniformly at random from $\mathcal{H}^{\mathrm{OMS}}_\alpha$,*

$$\mathbb{E}\delta_a(x,y) = \begin{cases} 0 & \text{if } i \ge u-s \\ 2/2^s & \text{if } i+|\alpha| \le u-s \\ \frac{1}{2^{u-|\alpha|-i}}\mathbb{1}\left\{\alpha(x-y) \bmod 2^{|\alpha|+i} \in M_{2^{u-s},2^{|\alpha|+i}}\right\} & \text{if } |\alpha|+i \le u \\ \mathbb{1}\left\{\alpha(x-y) \bmod 2^u \in M_{2^{u-s},2^u}\right\} & \text{if } |\alpha|+i \ge u \end{cases}.$$

Proof. When $i \ge u-s$, $\delta_a(x,y)=0$ by definition. Henceforth assume $i < u-s$. We can decompose a into its fixed lower bits α and uniformly random upper bits $a' \in \left[2^{u-|\alpha|}\right)$, so that $a = a'2^{|\alpha|}+\alpha$. Let $t = \min(|\alpha|+i,u)$ Then

$$a(x-y) \bmod 2^u = \left((a'z \bmod 2^{u-t})2^t + \alpha z 2^i\right) \bmod 2^u.$$

Since z is odd, and a' is chosen uniformly at random from $\left[2^{u-|\alpha|}\right)$, the product $a'z \bmod 2^{u-t}$ is distributed uniformly over $[2^{u-t})$. Let S be the set of possible values of $a(x-y) \bmod 2^u$.

Depending on $i+|\alpha|$, we have two cases:

1. If $i+|\alpha| \le u-s$, let $T = M_{2^{u-s},2^u} \cup \{2^{u-s}\}$. The set T is, $\bmod 2^u$, a contiguous interval containing 2^{u-s+1} elements. Since $i+|\alpha| \le u-s+1 \le u$, by Lemma 2.2, S has exactly $2^{u-s+1}/2^{i+|\alpha|}$ intersections with T. Because both α and z are odd and $|\alpha| \ge 1$, $a(x-y) \bmod 2^{i+1} = 2^i$. Because $i+1 \le u-s$, we have $2^{u-s} \bmod 2^{i+1} = 0$, which implies that the set S cannot contain 2^{u-s}. Thus $|S \cap M_{2^{u-s},2^u}| = |S \cap T|$. Since $a(x-y) \bmod 2^u$ is uniformly distributed over values in S, it follows

$$\Pr[a(x-y) \bmod 2^u \in M_{2^{u-s},2^u}] = \frac{2^{u-s+1}}{2^{i+|\alpha|}2^{u-|\alpha|-i}} = \frac{2}{2^s}.$$

2. If $i+|\alpha| > u-s$, then because the elements S are spaced 2^t apart, and $2^t \ge 2^{u-s+1}$, S has at most one intersection with $M_{2^{u-s},2^u}$. In particular, S intersects $M_{2^{u-s},2^u}$ iff $\alpha(x-y) \bmod 2^t \in M_{2^{u-s},2^t}$. That the latter condition is necessary follows because the only possible value of $a(x-y) \bmod 2^t$ is $\alpha(x-y) \bmod 2^t$; that it is sufficient follows because every translation of $\alpha(x-y) \bmod 2^t$ by a multiple of 2^t

is contained in S. Since there are 2^{u-t} values in S, the expected value if $S \cap M \neq \emptyset$ is $\frac{1}{2^{u-t}}$. [**This is a bit vague, and in general the entire proof needs cleanup.**] !!!

$\square$

Finally, we note that $\delta_a(x,y)$ is not much larger than and can sometimes be used instead of the actual collision probability function.

Corollary 2.5. *With h_a chosen uniformly at random from the set $\mathcal{H}^{\mathrm{OMS}}$ of OMS hash functions from $\{0,1\}^u$ to $\{0,1\}^s$, and x,y distinct values in $\{0,1\}^u$,*

$$\mathbb{E}\delta_a(x,y) \leq \frac{2}{2^s}.$$

Proof. Decompose $x-y = z2^i$ for odd z and integral i. Since either $i \geq u-s$, or $i+|\alpha| = i+1 \leq u-s$, when applying apply Lemma 2.4 we get an upper bound of either 0 or $2/2^s$. $\square$

3 Properties of the odd-multiply-add-shift (OMA) hash family

Define $\operatorname{mabs}(x,m) := \min(x \bmod m, -x \bmod m)$; this can be seen as a modular absolute value. Then for any v,w, $\operatorname{mabs}(v-w,m) = \operatorname{mabs}(w-v,m)$ is the minimum absolute value of an offset $d \in ZZ$ so that $(v+d) \bmod m = w \bmod m$.

Lemma 3.1 (Effect of a random offset on hash function collisions)**.** *Fix $u \geq s \geq 1$. Let w,v be (possibly identical) integers in $\{0,1\}^u$, and b a uniformly random integer in $[2^{u-s})$. Then:*

$$\Pr[w+b \bmod 2^u \operatorname{div} 2^{u-s} = v+b \bmod 2^u \operatorname{div} 2^{u-s}] = \frac{1}{2^{u-s}} \max\left(0, 2^{u-s} - \operatorname{mabs}(v-w, 2^{u-s})\right)$$

Proof. Observe that if, for some x,y, $x \operatorname{div} 2^{u-s} = y \operatorname{div} 2^{u-s}$, then $\operatorname{mabs}(x-y, 2^{u-s}) \leq |x-y| \leq 2^{u-s}-1$. Since $\operatorname{mabs}(v-w, 2^{u-s}) = \operatorname{mabs}((v+b)-(w+b), 2^{u-s})$ for any value of b, it follows that if $\operatorname{mabs}(v-w,2^{u-s}) \geq 2^{u-s}$, then

$$\Pr[w+b \bmod 2^u \operatorname{div} 2^{u-s} = v+b \bmod 2^u \operatorname{div} 2^{u-s}].$$

Consider the case where $\operatorname{mabs}(v-w,2^{u-s}) < 2^{u-s}$. Without loss of generality assume $v-w \bmod 2^{u-s} < 2^{u-s}$. We note that exactly $2^{u-s} - \operatorname{mabs}(v-w,2^{u-s})$ of the possible values of $b \in [2^{u-s})$ will place $(w+b) \bmod 2^{u-s}$ in the set $[2^{u-s} - \operatorname{mabs}(v-w,2^{u-s}))$, in which case $(v+b) \bmod 2^u \operatorname{div} 2^{u-s}$ will equal $(w+b) \bmod 2^u \operatorname{div} 2^{u-s}$. [**This is skipping much of the actual *proof*. One approach: use casework, like for the offset collision estimator function?**] !!! $\square$

Lemma 3.2. *The OMA hash family $\mathcal{H}^{\mathrm{OMA}}$ from $\{0,1\}^u$ to $\{0,1\}^s$ is universal.*

This will follow as a special case of the upcoming Lemma 3.3.

For odd $\alpha \in \{0,1\}^\star$, define $\mathcal{H}^{\mathrm{OMA}}_{\alpha,\star}$ to be the set of OMA hash functions with multiplier a whose least significant bits are α, and arbitrary offset. Formally,

$$\mathcal{H}^{\mathrm{OMA}}_{\alpha,\star} := \left\{ h_{a'2^{|\alpha|}+\alpha, b} : a' \in \left[2^{u-|\alpha|}\right), b \in \left[2^{u-s}\right) \right\}$$

Unlike for the OMS functions, we do not need any upper bound for the collision probability when hash functions are chosen uniformly from $\mathcal{H}^{\mathrm{OMA}}_{\alpha,\star}$, because the exact value can be computed directly.

[**In general lemmas, $[2^u)$ might be preferable to $\{0,1\}^u$?**] !!!

[4] These properties follow from that fact that the multiplicative group $(\mathbb{Z}/2^k\mathbb{Z})^\times$ of $\mathbb{Z}/2^k\mathbb{Z}$ consists of the equivalence classes of odd integers in integers in $[2^k)$.

[5] Since a is odd, $ax \bmod 2^u$ will never equal $ay \bmod 2^u$, and these two subcases are disjoint; but this is not critical to the proof.

Lemma 3.3. *Let $u \geq s \geq 1$. For distinct $x, y \in \{0,1\}^u$, let z, i be integers for which z is odd and $x - y \bmod 2^u = z2^i$. Fix odd $\alpha \in \{0,1\}^\star$. If $h_{a,b}$ is chosen uniformly at random from $\mathcal{H}^{\mathrm{OMA}}_{\alpha,\star}$, then:*

$$\Pr[h_{a,b}(x) = h_{a,b}(y)] = \begin{cases} 0 & \text{if } i \geq u-s \\ \frac{1}{2^s} & \text{if } |\alpha| + i \leq u-s \\ \frac{1}{2^{2u-s-|\alpha|-i}} \max\left(0, 2^{u-s} - \mathrm{mabs}(\alpha x - \alpha y, 2^{|\alpha|+i})\right) & \text{if } u \geq |\alpha| + i > u-s \\ \frac{1}{2^{u-s}} \max\left(0, 2^{u-s} - \mathrm{mabs}(\alpha x - \alpha y, 2^u)\right) & \text{if } |\alpha| + i \geq u \end{cases}.$$

Proof. First, consider the case where $i \geq u - s$. Then

$$(ax - ay) \bmod 2^{u-s} = z2^i \bmod 2^{u-s} = 0\,. \tag{3}$$

Since a is always odd, the map $z \mapsto az \bmod 2^u$ is always a bijection, so $\mathrm{mabs}(ax - ay, 2^u)$ cannot be zero and must instead be $\geq 2^{u-s}$. Applying Lemma 3.1 with $w = ax$ and $v = ay$ implies $\Pr[h_{a,b}(x) = h_{a,b}(y)] = 0$.

For the remaining cases, we have $i < u - s$. By the definition of $\mathcal{H}^{\mathrm{OMA}}_{\alpha,\star}$, the random variable $a = a'2^{|\alpha|} + \alpha$ for an a' distributed uniformly randomly over $\left[2^{u-|\alpha|}\right)$.

For a given value of a', the probability (over b) that $h_{a,b}(x) = h_{a,b}(y)$ is, by Lemma 3.1, a function of

$$(ax - ay) \bmod 2^u = (a'z2^{|\alpha|+i} + \alpha z 2^i) \bmod 2^u$$

Since z is odd, and a' is chosen uniformly at random over $\left[2^{u-|\alpha|}\right)$, their product $a'z \bmod 2^{u-|\alpha|-i}$ is uniformly distributed over $\left[2^{u-|\alpha|-i}\right)$. Also, since α is odd, αz is odd.

Let S be the set of possible values of $ax - ay \bmod 2^u$. These values differ ($\bmod 2^u$) by multiples of $2^{|\alpha|+i}$, if $|\alpha| + i < u$; if $|\alpha| + i \geq u$, then S contains only $\alpha z 2^i \bmod 2^u$.

The probability from Lemma 3.1 is a function of $ax - ay \bmod 2^u$ which is supported on $M_{2^{u-s},2^u}$. Define $T = M_{2^{u-s},2^u} \cup \{2^{u-s}\}$

We have two main cases:

1. If $|\alpha| + i \leq u - s$, then $|S \cap T| \geq 2$, because T forms (in $\mathbb{Z}/2^u\mathbb{Z}$) an interval of length 2^{u-s+1} and by Lemma 2.2 has $2^{u-s+1}/2^{|\alpha|+i}$ intersections with S. Furthermore, we can match up intersections of S with $T_- := \{2^u - 2^{u-s} + 1, \ldots, 2^u - 1\} \cup \{0\}$ with intersections of S with $T_+ := \{1, \ldots, 2^{u-s}\}$, matching values $r \in T_+$ with $r - 2^{u-s} \bmod 2^u$ in T_-. Applying Lemma 3.1 for each value in S, we obtain:

$$\begin{aligned} \Pr[h_{a,b}(x) = h_{a,b}(y)] &= \frac{1}{|S|2^{u-s}} \sum_{r \in S} \max(0, 2^{u-s} - \mathrm{mabs}(r, 2^u)) \\ &= \frac{1}{2^{u-|\alpha|-i}2^{u-s}} \sum_{r \in T} (2^{u-s} - \mathrm{mabs}(r, 2^u)) \\ &= \frac{1}{2^{u-|\alpha|-i}2^{u-s}} \sum_{r \in T_+} ((2^{u-s} - \mathrm{mabs}(r, 2^u)) + (2^{u-s} - \mathrm{mabs}((r - 2^{u-s}) \bmod 2^u, 2^u))) \\ &= \frac{1}{2^{u-|\alpha|-i}2^{u-s}} \sum_{r \in T_+} ((2^{u-s} - r) + (2^{u-s} - (2^{u-s} - r))) \\ &= \frac{1}{2^{u-|\alpha|-i}2^{u-s}} \frac{2^{u-s}}{2^{|\alpha|+i}} 2^{u-s} = \frac{1}{2^s}\,. \end{aligned}$$

   (See Figure 1 for a diagram of the values of $\max(0, 2^{u-s} - \mathrm{mabs}(r, 2^u))$ on S.)

2. If $|\alpha| + i > u - s$, then $|S \cap M_{2^{u-s},2^u}| \leq 1$, again by Lemma 2.2. For $|\alpha| + i < u$, the absolute distance of the closest value in S to 0 is preserved when taking all elements in S, $\bmod 2^{|\alpha|+i}$, and is $\mathrm{mabs}(\alpha x -$

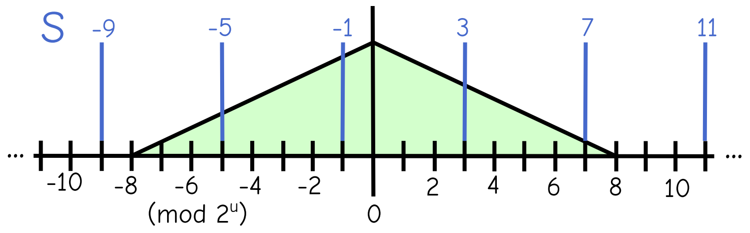

Figure 1: Evaluation of the function $\max(0, 2^{u-s} - \operatorname{mabs}(r, 2^u))$ (shown in green) over the set S (blue), from the proof of Lemma 3.3, for example inputs.

$\alpha y, 2^{|\alpha|+i})$. **[needs better explanation]**. If any element of S yield a nonzero collision probability, it will be this one. Similarly, if $|\alpha| + i \geq u$, the collision probability for the only possible value of $ax - ay \bmod 2^u$ in S is a function of $\operatorname{mabs}(\alpha x - \alpha y, 2^u)$. Applying Lemma 3.1, we get: !!!

$$\begin{aligned}
\Pr[h_{a,b}(x) = h_{a,b}(y)] &= \frac{1}{|S|} \cdot \frac{1}{2^{u-s}} \max(0, 2^{u-s} - \operatorname{mabs}(\alpha x - \alpha y, \min(2^u, 2^{|\alpha|+i}))) \\
&= \frac{1}{2^{\max(0, u-|\alpha|-i)}} \cdot \frac{1}{2^{u-s}} \max(0, 2^{u-s} - \operatorname{mabs}(\alpha x - \alpha y, \min(2^u, 2^{|\alpha|+i}))) .
\end{aligned}$$

□

Proof of Lemma 3.2. When $\alpha = (1)$, $\mathcal{H}^{\text{OMA}}_{\alpha,\star} = \mathcal{H}^{\text{OMA}}$. Let x, y be any distinct values in $\{0,1\}^u$, and let z, i be integers so that z is odd and $x - y \bmod 2^u = z2^i$. Now apply the collision bound of Lemma 3.3. Since $|\alpha| = 1$, either $i \geq u - s$ or $|\alpha| + i = i + 1 \leq u - s$; in both cases the probability that $h_{a,b}(x) = h_{a,b}(y)$ is $\leq 1/2^s$. □

For each $\beta \in \{0,1\}^\star$, define $\mathcal{H}^{\text{OMA}}_{a,\beta}$ to be the set of OMA hash functions with multiplier a and whose offset's highest $|\beta|$ bits match β; that is,

$$\mathcal{H}^{\text{OMA}}_{a,\beta} := \left\{ h_{a,\beta 2^{u-s-|\beta|}+b'} : b \in \left[2^{u-s-|\beta|}\right) \right\}$$

Lemma 3.4. *Say x, y are distinct values in $\{0,1\}^u$. If $h_{a,b}$ is chosen uniformly at random from $\mathcal{H}^{\text{OMA}}_{a,\beta}$:*

$$\Pr[h_{a,b}(x) = h_{a,b}(y)] = \begin{cases} f(x,y) & \textit{if } (ay - ax) \bmod 2^u < 2^{u-s} \\ f(y,x) & \textit{if } (ax - ay) \bmod 2^u < 2^{u-s} \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

$$\begin{aligned}
\textit{where} \qquad f(x,y) &= \mathbb{1}\{(ay + \beta 2^{u-s-|\beta|}) \bmod 2^{u-s} \geq (ax + \beta 2^{u-s-|\beta|}) \bmod 2^{u-s}\} \\
&\quad + \frac{\sigma((ax + \beta 2^{u-s-|\beta|})) - \sigma((ay + \beta 2^{u-s-|\beta|}))}{2^{u-s-|\beta|}} \\
\textit{where} \qquad \sigma(w) &= \max(0, w \bmod 2^{u-s} - (2^{u-s} - 2^{u-s-|\beta|}))
\end{aligned}$$

Proof. Since $h_{a,b}$ is drawn from $\mathcal{H}^{\text{OMA}}_{a,\beta}$, $b = \beta 2^{u-s-|\beta|} + b'$ where b' is chosen uniformly at random from $\left[2^{u-s-|\beta|}\right)$.

For $(ax + b) \bmod 2^u \operatorname{div} 2^{u-s}$ to equal $(ay + b) \bmod 2^u \operatorname{div} 2^{u-s}$, both $(ax + b) \bmod 2^u$ and $(ay + b) \bmod 2^u$ must lie in the same set $\{k2^{u-s}, \ldots, k2^{u-s} + 2^{u-s} - 1\}$ for some integer k. This can *only* happen if either $(ax - ay) \bmod 2^u < 2^{u-s}$ or $(ay - ax) \bmod 2^u < 2^{u-s}$; if neither of these two conditions hold, $(ax + b) \bmod 2^u \operatorname{div} 2^{u-s}$ and $(ay + b) \bmod 2^u \operatorname{div} 2^{u-s}$ will be different, no matter what b is.

We now evaluate the case where $(ay - ax) \bmod 2^u < 2^{u-s}$ holds; by symmetry, the probability that $h_{a,b}(x) = h_{a,b}(y)$ will have a similar form (with x and y swapped) when $(ax - ay) \bmod 2^u < 2^{u-s}$.

For a given value $w \in [2^u)$, the number of values $b' \in \left[2^{u-|\beta|}\right)$ for which $(w + b') \bmod 2^u \operatorname{div} 2^{u-s} \neq w \bmod 2^u \operatorname{div} 2^{u-s}$ is exactly

$$\sigma(w) = \max(0, w \bmod 2^{u-s} - (2^{u-s} - 2^{u-s-|\beta|}))$$

since $(w+b') \bmod 2^u$ only enters the interval of 2^{u-s} elements mapping to $(w \operatorname{div} 2^{u-s} + 1) \bmod 2^s$ if w mod 2^{u-s} is close enough to 2^u for the addition to b' to make $(w \bmod 2^{u-s}) + b' \geq 2^{u-s}$.

Because $(ay - ax) \bmod 2^u < 2^{u-s}$, we have two cases:

- $h_{a,b}(x)$ and $h_{a,b}(y)$ collide when $b' = 0$. In this case,

  $$(ay + \beta 2^{u-s-|\beta|}) \bmod 2^{u-s} \geq (ax + \beta 2^{u-s-|\beta|}) \bmod 2^{u-s}$$

  since $(ax + \beta 2^{u-s-|\beta|}) \bmod 2^u$ and $(ay + \beta 2^{u-s-|\beta|}) \bmod 2^u$ are in the same coset of $[2^{u-s})$ and taking $\bmod 2^{u-s}$ does not change their order. As b' increases from 0 to $2^{u-s} - 1$, first $(ay + \beta 2^{u-s-|\beta|} + b') \operatorname{div} 2^{u-s}$ might increase (preventing a collision), and then $(ax + \beta 2^{u-s-|\beta|} + b') \operatorname{div} 2^{u-s}$ might increase (preserving a collision). The total probability that b' is such that $h_{a,b}(x) = h_{a,b}(y)$ is thus:

  $$\frac{1}{2^{u-s-|\beta|}}(2^{u-s-|\beta|} + \sigma((ax + \beta 2^{u-s-|\beta|})) - \sigma((ay + \beta 2^{u-s-|\beta|})))$$

- $h_{a,b}(y) = (h_{a,b}(y) + 1) \bmod 2^{u-s}$ when $b' = 0$. Since $(ax - ay) \bmod 2^u < 2^{u-s}$, in this case we have:

  $$(ay + \beta 2^{u-s-|\beta|}) \bmod 2^{u-s} < (ax + \beta 2^{u-s-|\beta|}) \bmod 2^{u-s}$$

  As b' is increased, first $(ax + \beta 2^{u-s-|\beta|} + b') \operatorname{div} 2^{u-s}$ will increase (creating a collision), followed by $(ay + \beta 2^{u-s-|\beta|} + b') \operatorname{div} 2^{u-s}$ (preventing a collision). Similarly to the first case:

  $$\Pr[h_{a,b}(x) = h_{a,b}(y)] = \frac{1}{2^{u-s-|\beta|}}(\sigma((ax + \beta 2^{u-s-|\beta|})) - \sigma((ay + \beta 2^{u-s-|\beta|})))$$

:

Summarizing all cases yields Eq. 4. □

4 Properties of the odd-carryless-multiply-shift (OCM) hash family

For odd $\alpha \in \{0,1\}^\star$, define $\mathcal{H}_\alpha^{\mathrm{OCM}}$ to be the set of OCM hash functions with multiplier a whose least significant bits are α. Formally,

$$\mathcal{H}_\alpha^{\mathrm{OCM}} := \left\{ h_{a'2^{|\alpha|}+\alpha,b} : a' \in \left[2^{u-|\alpha|}\right) \right\}$$

Lemma 4.1. *Fix some nonzero $w \in \{0,1\}^u$ with $i := \textsc{tzcnt}(w)$. Let $j \geq 1$, $k \geq 0$ be integers with $j + k + i \leq u$. Given odd $\alpha \in \{0,1\}^j$, then for every $z \in \{0,1\}^k$, there exists a unique $a' \in \{0,1\}^k$ so that*

$$((\alpha + a'2^j) \odot w) \operatorname{sel} [i + j, i + j + k) = z.$$

Proof. We describe how to construct a'. Let $a'_0.a'_1 \ldots a'_{k-1}$ be the bits of a'. For each $h \in [0,k)$, in ascending order, note that

$$\begin{aligned}((\alpha + 2^j a') \odot w)_{i+j+h} &= (\alpha \odot w)_{i+j+h} \oplus \bigoplus_{g=0}^{k-1} ((2^{j+g} a'_g) \odot w)_{i+j+h} \\ &= (\alpha \odot w)_{i+j+h} \oplus \bigoplus_{g=0}^{h} (a'_g w_{i+h-g})\end{aligned}$$

since the bits of w at positions below i are all zero. As $w_i = 1$, setting

$$a'_h = z_h \oplus ((\alpha + 2^j(a' \bmod 2^h)) \odot w)_{i+j+h}$$

will ensure $((\alpha + 2^j a') \odot w)_{i+j+h}$ matches z_h without affecting the result for any smaller values of h. □

Lemma 4.2. *Say x,y are distinct values in $\{0,1\}^u$. Let i,z be integers for which z is odd and $x - y = z2^i$. Given some odd $\alpha \in \{0,1\}^\star$, if h_a is chosen uniformly at random from $\mathcal{H}_\alpha^{\mathrm{OCM}}$:*

$$\Pr[h_a(x) = h_a(y)] = \begin{cases} 0 & \text{if } i \geq u - s \\ \frac{1}{2^s} & \text{if } |\alpha| + i \leq u - s \\ \frac{1}{2^{u-|\alpha|-i}} & \text{if } (\alpha \odot (x \oplus y)) \operatorname{sel} [u-s, |\alpha| + i) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Proof. Note that i indicates the least significant bit of $x \oplus y$. If $i \geq u - s$, then by linearity of carryless multiplication, $(a \odot x) \oplus (a \odot y) = a \odot (x \oplus y)$. Since a is odd, the least significant nonzero bit of a value is preserved when multiplying by a. Consequently, the ith bit of $a \odot (x \oplus y)$ is set, so the $i - (u - s)$th bit of $h_a(x) \oplus h_a(y) = a \odot (x \oplus y) \operatorname{div} 2^{u-s}$ is set, which implies $h_a(x) \neq h_a(y)$.

If h_a is chosen uniformly at random from $\mathcal{H}_\alpha^{\mathrm{OCM}}$, then $a = \alpha + 2^{|\alpha|} a'$ where a' is distributed uniformly at random over $\left[2^{u-|\alpha|}\right)$. By Lemma 4.1, for each $z \in \left[2^{u-|\alpha|-i}\right)$, there is a unique value of $a' \bmod 2^{u-|\alpha|-i}$ for which $(\alpha + 2^{|\alpha|} a') \odot (x \oplus y) \operatorname{sel} [|\alpha| + i, u) = z$. This implies that there is a bijection between $(\alpha + 2^{|\alpha|} a') \odot (x \oplus y) \operatorname{sel} [|\alpha| + i, u)$ and $a' \bmod 2^{u-|\alpha|-i}$. Since $a' \bmod 2^{u-|\alpha|-i}$ is uniformly distributed, it follows that $(\alpha + 2^{|\alpha|} a') \odot (x \oplus y) \operatorname{sel} [|\alpha| + i, u)$ is uniformly distributed. On the other hand, we have:

$$\Big((\alpha + 2^{|\alpha|} a') \odot (x \oplus y)\Big) \operatorname{sel} [0, |\alpha| + i) = (\alpha \odot (x \oplus y)) \operatorname{sel} [0, |\alpha| + i).$$

Summarizing, the $|\alpha| + i$ least significant bits of $a \odot (x \oplus y) \bmod 2^u$ are fixed, and the $u - |\alpha| + i$ most significant bits of $a \odot (x \oplus y) \bmod 2^u$ are uniform.

Consequently, if $|\alpha| + i \leq u - s$, then $h_a(x) \oplus h_a(y)$ is uniformly distributed over $[2^s)$ and equals zero with exactly $1/2^s$ probability. If $\alpha + i > u - s$, then if $\alpha \odot (x \oplus y) \operatorname{sel} [u - s, \alpha + i)$ is nonzero, we will always have $h_a(x) \neq h_a(y)$; if $\alpha \odot (x \oplus y) \operatorname{sel} [u - s, \alpha + i)$ is zero, then $h_a(x) = h_a(y)$ iff $a \odot (x \oplus y) \operatorname{sel} [\alpha + i, u) = 0$ which occurs with probability $1/2^{u-|\alpha|-i}$. □

Corollary 4.3. *For any $u \geq s \geq 0$, the OCM hash family from $\{0,1\}^u$ to $\{0,1\}^s$ is universal.*

Proof. Say h_a is chosen uniformly from $\mathcal{H}^{\mathrm{OCM}}$. For any distinct $x,y \in \{0,1\}^u$, let $i = \textsc{tzcnt}(x \oplus y)$. By Lemma 4.2 with $\alpha = (1)$, either $i \geq u - s$, in which case $\Pr[h_a(x) = h_a(y)] = 0$, or $i + |\alpha| = i + 1 \leq u - s$, in which case $\Pr[h_a(x) = h_a(y)] = 1/2^s$. □

5 Faster selection of OMS hash functions

[Ram95] identified an odd-multiply-hash function with collision count not more than average using the method of conditional expectations in $O(n^2u)$ time. The expected value of the upper bound on the collision probability (Lemma 2.4) in fact can be efficiently evaluated in batches, ultimately leading to an $O(nu\log n)$ time algorithm.

Our goal is to select an OMS hash function multiplier a for which the sum of pairwise collision upper bounds,

$$\tau(a) = \sum_{\{x,y\}\in\binom{S}{2}} \delta_a(x,y) \tag{5}$$

is less than the expected value over all odd $a \in [2^b)$. Recall that $\delta_a(x,y)$ is an upper bound on $\mathbb{1}\{h_a(x) = h_a(y)\}$. Using Corollary 2.5, the selected value for a, $a_\star$, will satisfy

$$\frac{2}{2^s}\binom{n}{2} \geq \mathbb{E}[\tau(a)] \geq \tau(a_\star) \geq \sum_{\{x,y\}\in\binom{S}{2}} \mathbb{1}\{h_a(x) = h_a(y)\} = \kappa(h_{a_\star}, S)$$

so the chosen multiplier $a_\star$ also guarantees few collisions for S under $h_{a_\star}$.

To pick a good $a_\star$, we use the method of conditional expectations to identify acceptable bits of $a_\star$ in order from least significant to most significant, progressively extending an lsb-prefix $\alpha \in \{0,1\}^s tar$ of set bits while not increasing the expected value of $\tau(a)$ conditioned on random a being an extension of α. See for example Algorithm 1.

Algorithm 1 Bit-by-bit parameter selection for $\tau : \{0,1\}^u \to \mathbb{R}$ by the method of conditional expectations

```
1:  $\alpha \leftarrow (1)$
2:  $q \leftarrow \mathbb{E}[\tau(a)|a$ extends $\alpha]$
3:  for $k$ in $\{1,\ldots,u-1\}$ do
4:      $p \leftarrow \mathbb{E}[\tau(a)|a$ extends $\alpha.(0)]$
5:      if $p \leq q$ then
6:          $\alpha \leftarrow \alpha.(0)$
7:          $q \leftarrow p$
8:      else
9:          $\alpha \leftarrow \alpha.(1)$
10:         $q \leftarrow 2q - p$
11: return $\alpha$
```

The key observation in this section is that τ can be efficiently computed by evaluating the conditional expectations of $\delta_a(x,y)$ over certain batches of pairs $\{x,y\}$. Define, for each $i \in \{0,\ldots,u-1\}$, $\gamma \in \{0,1\}^i$, and t in $\{0,1\}$, the set $S_{i,\gamma,t}$ to contain all elements in S whose $(i+1)$ least significant bits are $\gamma.(t)$. (Equivalently, $x \in S_{i,\gamma,t}$ iff $x \bmod 2^{i+1} = \sum_{j=0}^{i} \gamma_j 2^j + t2^i$.) Since every pair $\{x,y\} \in \binom{S}{2}$ shares some prefix length i of least significant bits, differing on the next (ith, counting from 0) bit:

$$\binom{S}{2} = \bigcup_{i\in\{0,\ldots,u-1\}} \bigcup_{\gamma\in\{0,1\}^i} S_{i,\gamma,0} \times S_{i,\gamma,1}$$

Matching this decomposition, we can evaluate, for any α, $\mathbb{E}[\tau(a)|a$ extends $\alpha]$ as a sum of terms:

$$\phi(\alpha,i,\gamma) := \sum_{x\in S_{i,\gamma,0}} \sum_{y\in S_{i,\gamma,1}} \mathbb{E}[\delta_a(x,y)|a \text{ extends } \alpha]$$

By Lemma 2.4, for any x, y, the quantity $\mathbb{E}[\delta_a(x,y) | a \text{ extends } \alpha]$ evaluates to a piecewise-defined function, whose branches depend on $|\alpha|$ and $i := \textsc{tzcnt}(x \oplus y)$. When evaluating $\phi(\alpha, i, \gamma)$, these parameters are constant and we only need to consider a single branch. There are three main *regions* for a given $|\alpha|$, depending on i:

trivial region If $i \geq u - s$, then for all $x \in S_{i,\gamma,0}, y \in S_{i,\gamma,1}$, $\mathbb{E}\delta_a(x,y) = 0$, so $\phi(\alpha, i, \gamma) = 0$.

base region If $i + |\alpha| \leq u - s$, then for all $x \in S_{i,\gamma,0}, y \in S_{i,\gamma,1}$, $\mathbb{E}\delta_a(x,y)$ is constant; thus $\phi(\alpha, i, \gamma) = |S_{i,\gamma,0}| \cdot |S_{i,\gamma,1}| \cdot 2/2^s$.

interactive region The remaining case, where $i < u - s$ and $i + |\alpha| > u - s$, is trickier, but assuming $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$ are provided as lists in a certain order, Lemma 5.1 describes how to compute $\phi(\alpha, i, \gamma)$ efficiently.

Lemma 5.1. *Fix some odd $\alpha \in \{0,1\}^\star$, $i \in \{0, \ldots, u-1\}$, and $\gamma \in \{0,1\}^i$. Say $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$ are provided as lists that have been sorted by the key function $x \mapsto \alpha x \bmod 2^{\min(|\alpha|+i,u)}$. If $i < u - s$ and $i + |\alpha| > u - s$, then we can evaluate $\phi(\alpha, i, \gamma)$ in linear ($O(|S_{i,\gamma,0}| + |S_{i,\gamma,1}|)$) time.*

Proof. [**special case, if values are not all distinct.**] !!!

For any $x \in S_{i,\gamma,0}$, the set of $y \in S_{i,\gamma,1}$ for which [**specific condition holds**] forms a contiguous (albeit possibly wrapping around) interval $I_x \subseteq S_{i,\gamma,1}$. Using Algorithm 2, we can efficiently identify this interval for a given x, update it as we iterate over values in $S_{i,\gamma,0}$, and calculate the total number of (x,y) pairs satisfying [**objective**]; this is just $p = \sum_{x \in S_{i,\gamma,0}} |I_x|$. Then $\phi(\alpha, i, \gamma) = p \cdot ???$. !!! !!!

In general, have a "Sliding window" calculation, which is slightly complicated by wraparound. !!!

[**Sublemma: L/R, after being set initially, walk around S_1 exactly once. Maybe create a figure showing, on a common or parallel circles, the evaluation window and the points.**] !!!

□

5.1 Fast evaluation of the $i + |\alpha| \leq u - s$ case

We will now describe how to efficiently compute, for all $i \in \{0, \ldots, u-1\}$, the quantities

$$\pi(i) := \sum_{\gamma \in \{0,1\}^i} |S_{i,\gamma,0}| \cdot |S_{i,\gamma,1}| \tag{6}$$

When $i + |\alpha| \leq u - s$, the quantity $\sum_{\gamma \in \{0,1\}^i} \phi(\alpha, i, \gamma) = \frac{2}{2^s}\pi(i)$. This is one of the components to computing $\tau(\alpha)$.

First, sort the input S in ascending order using a *bit-reversed* comparison, ordering x before y iff $\textsc{brev}(x) < \textsc{brev}(y)$. This order groups together values sharing the least significant bits; see for example Figure 2.

The values $(\pi(i))_{i \in \{0,\ldots,u-1\}}$ can be computed in a single pass over the sorted input, using a $\leq u$-height stack to keep track of when the last batch of values sharing some number of least significant bits begins. An algorithm for this is given by Algorithm 3. The key idea here is that, for some i, γ pair, the sets $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$ if nonempty are contiguous and adjacent to each other, with $S_{i,\gamma,1}$ following $S_{i,\gamma,0}$. The start of $S_{i,\gamma,0}$ and end of $S_{i,\gamma,1}$ are both marked by a transition of a bit at position k for some $k < i$ from 0 to 1; since each element x in $S_{i,\gamma,0} \sqcup S_{i,\gamma,1}$ has $x \operatorname{sel} [0, i) = \gamma$, there will be no such transitions between the start of $S_{i,\gamma,0}$ and end of $S_{i,\gamma,1}$. The index at which $S_{i,\gamma,0}$ ends and $S_{i,\gamma,1}$ starts is marked with the ith bit of the elements changing from 0 to 1; if both $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$ are nonempty, this occurs exactly once. Consequently, each nonzero $|S_{i,\gamma,0}||S_{i,\gamma,1}|$ term corresponds to a bit transition of the ith bit in the sorted input, and the values of $|S_{i,\gamma,0}|$ and $|S_{i,\gamma,1}|$ can be determined by looking at the preceding and following transitions for lower bits.

[**is a detailed correctness proof necessary? Also, runtime estimation.**] !!!

Algorithm 2 Efficient evaluation of $\phi(\alpha, i, \gamma)$ in the interactive region.

```
    $t \leftarrow \min(u, |\alpha| + i)$.                                          ▷ Note that $t > u - s$

1:  function BatchPairEstimate($S_0$, $S_1$)     ▷ Requires: $S_0$ and $S_1$ to be sorted by key $x \mapsto (\alpha x) \bmod 2^t$
2:      if $|S_0| \leq 1$ or all $\alpha y \bmod 2^t$ for $y \in S_1$ are equal then
3:          $p \leftarrow 0$
4:          for $x$ in $S_0$, any order do
5:              if $(\alpha(x - S_1[0]) - 2^{u-s} + 1) \bmod 2^t \leq 2^{u-s+1} - 1$ then
6:                  $p \leftarrow p + 1$
7:
8:          return $p \cdot \frac{|S_1|}{2^{u-t}}$
9:      $p \leftarrow 0$
10:     $L \leftarrow 0$                                                   ▷ Any initial value for L and R will work
11:     $R \leftarrow 0$
12:     for $x$ in $S_0$, in order do
13:         ▷ Update L, R to be rightmost indices minimizing $(T_L - \alpha S_1[L]) \bmod 2^t$ and similarly for R ◁
14:         $T_L \leftarrow (\alpha x - 2^{u-s}) \bmod 2^t$
15:         $T_R \leftarrow (\alpha x + 2^{u-s} - 1) \bmod 2^t$
16:         while $(T_L - S_1[(L+1) \bmod |S_1|]) \bmod 2^t \leq (T_L - \alpha S_1[L]) \bmod 2^t$ do
17:             $L = (L+1) \bmod |S_1|$
18:         while $(T_R - S_1[(R+1) \bmod |S_1|]) \bmod 2^t \leq (T_R - \alpha S_1[R]) \bmod 2^t$ do
19:             $R = (R+1) \bmod |S_1|$
20:         if $L \neq R$ then
21:             $p \leftarrow p + ((R - L) \bmod |S_1|)$      ▷ Only $S_1[j]$ for $j$ in the $(L, R]$ modular interval match
22:         else if $(\alpha(x - S_1[0]) - 2^{u-s} + 1) \bmod 2^t \leq 2^{u-s+1} - 1$ then
23:             $p \leftarrow p + |S_1|$                                     ▷ All $y \in S_1$ match
24:         else
25:             ▷ No matching $y \in S_1$ ◁
26:     return $p \cdot \frac{1}{2^{u-t}}$
```

5.2 Aggregation and sorting

A single pass scan as in Algorithm 3 can be used to identify all pairs of nonempty sets $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$. To handle the "interactive region" of i parameters for a given α, we could immediately sort $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$ as needed and then apply Algorithm 2 to each. It is possible to avoid some work with the following observations:

Observation 5.2. *For any i, γ, b, the set $S_{i,\gamma,b}$ can be decomposed as $S_{i+1,\gamma.(b),0} \sqcup S_{i+1,\gamma.(b),1}$. If one has already sorted $S_{i+1,\gamma.(b),0}$ and $S_{i+1,\gamma.(b),1}$ by key function $x \mapsto (\alpha x) \bmod 2^{t'}$ for $t' = \min(|\alpha| + (i+1), u)$, then from these sorted sets one can compute the sorted version of $S_{i,\gamma,0}$, by key function $x \mapsto (\alpha x) \bmod 2^t$ for $t = \min(|\alpha| + (i+1), u)$, in $O(|S_{i,\gamma,0}|)$ time. (See* FilterMergeStep *of Algorithm 4, and Figure 3 for an example.)*

Observation 5.3. *When $i + |\alpha|$ grows above u, the value of $\phi(\alpha, i, \gamma)$ stops changing; this reduces the amount of computation needed. Specifically, [**Precise conditions and proof here**].*

*[**changing higher bits does not affect the value of** $\alpha(x - y) \bmod \ldots$ **/ of** $\phi(\alpha, i, \gamma)$**.**]* !!! !!!

An algorithm to compute $\mathbb{E}[\tau(a) | h_a \in \mathcal{H}_\alpha^{\text{OMS}}]$ which applies these two observations is given in Algorithm 4. Define $\rho(\alpha, i) = \sum_{\gamma \in \{0,1\}^i} \phi(\alpha, i, \gamma)$.

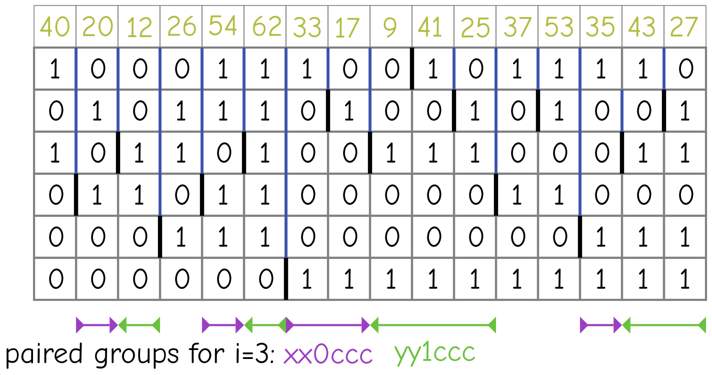

Figure 2: Sorting an array of integers (top) by their bit-reversed forms (middle) naturally groups together values which share the least significant i bits. The ”levels” of transitions from 0 to 1 are shown as short bold lines. The bottom of the diagram shows nonempty pairs of sets of values that match on the least significant $i = 3$ bits but differ on the next bit.

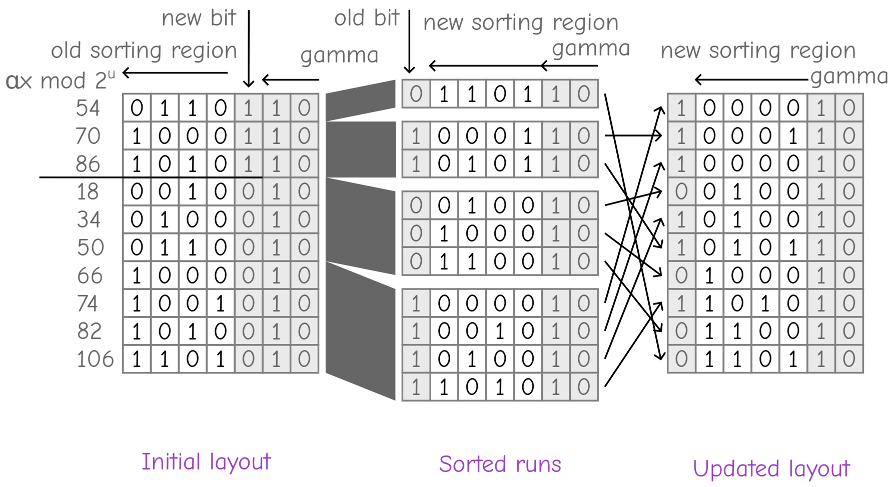

Figure 3: The filter-merge step of Algorithm 4 on example input, with $\gamma = (0,1)$, $i = 2$, $s = 4$, $u = 7$. Note: the bit strings here show the values of $(\alpha x \bmod 2^u)_{x \in S}$, *not* $(x)_{x \in S}$.

5.3 Implementation and runtime

All probabilities and expected values calculates in this algorithm are multiples of $1/2^u$, and the maximum expected value is $O(\binom{n}{2}/2^s)$, so only $O(\log(n) + u)$ bits are required to encode the various expected values computed.

The total runtime of the algorithm in the (word-RAM?) model is $O(nu(s + \log n))$. [**prove and elaborate**] !!!

Algorithm 3 Fast calculation of batch pair size products ($\pi(i)$ from Eq. 6), given a pre-sorted input.

```
1: function ProcessStackDownTo($i$, $j$, $\pi$ : &$mut$, $P$ : &$mut$)  ▷ $\pi$ is an array in $\mathbb{N}^u$, $P$ a stack of pairs of integers
2:     while $P$ is nonempty do
3:         Let $(j_0, i_0) \leftarrow$ top element of $P$
4:         if $i_0 < i$ then return
5:         Remove the top element of $P$
6:         if $P$ is nonempty then
7:             $(j_1, i_1) \leftarrow$ (the new) top element of $P$
8:         else
9:             $j_1 \leftarrow 0$
10:        $\pi[i_0] \leftarrow \pi[i_0] + (j - j_0) \cdot (j_0 - j_1)$

11: function ComputeBatchPairs($S$)  ▷ Requires: $S$ to already be sorted by bit-reversed values
12:     $\pi \leftarrow (0, \ldots, 0) \in \mathbb{N}^u$
13:     $P \leftarrow$ empty stack of (S-index, bit-index) pairs
14:     for $j \in \{1, \ldots, n-1\}$ do
15:         $i \leftarrow \text{TZCNT}(S[j-1] \oplus S[j])$
16:         ProcessStackDownTo($i$, &$mut\,\pi$, &$mut\,P$)
17:         Push $(j, i)$ onto $P$
18:     ProcessStackDownTo(0, &$mut\,\pi$, &$mut\,P$)
19:     return $\pi$
```

Algorithm 4 Evaluation of $\mathbb{E}[\tau(a)|a \text{ extends } \alpha]$, given preceding values

```
1: function FILTERMERGESTEP(L: &mut, i)
2:     $t = i + |\alpha|$
3:     Split $L$ into lists $L_0$ and $L_1$ so that $L_b := \{x \in L : (\alpha x) \operatorname{div} 2^{i+2} \bmod 2 = b\}$.
4:     if $t \geq u$ then
5:         $L \leftarrow$ sorted merge of $L_0$ and $L_1$ by key function $x \mapsto (\alpha x) \bmod 2^u$
6:     else
7:         Split $L_0$,$L_1$ into $L_{0,0}$,$L_{0,1}$, and $L_{1,0}$, $L_{0,1}$ so that $L_{b,c} := \{x \in L_b : (\alpha x) \operatorname{div} 2^t \bmod 2 = c\}$.
8:         $L \leftarrow$ sorted merge of $L_{0,0}$,$L_{0,1}$,$L_{1,0}$, and $L_{1,1}$ by key function $x \mapsto (\alpha x) \bmod 2^t$

9: ▷ $S \subseteq \{0,1\}^u$ is the set of values; $\sigma \in \mathbb{R}^{u-s}$ the probabilities by i-level(todo, def) from the preceding
   evaluation for $\alpha'.(0)$ so that $\alpha = \alpha'.(b).(0)$ for some bit b.  ◁
10: function EXPECTEDTAU($S$, $\sigma$, $\alpha$)
11:     $\rho \leftarrow (0, \dots, 0) \in \mathbb{R}^u$
12:     Sort $S$ ascending by key function $x \mapsto \text{BREV}(x)$
13:     $\pi \leftarrow$ COMPUTEBATCHPAIRS($S$)                                ▷ From Algorithm 3
14:     ▷ Base region                                                        ◁
15:     for $i \in \{0, \dots, u - s - |\alpha|\}$ do
16:         $\rho[i] \leftarrow \pi[i] \cdot 2/2^s$
17:     ▷ Unchanged values in interactive region: TODO explain boundary condition  ◁
18:     for $i \in \{\dots, \min(2 + u - s - |\alpha|, u - s), u - s - 1\}$ do
19:         $\rho[i] \leftarrow \sigma[i]$
20:     ▷ Interactive region                                                 ◁
21:     first ← true
22:     for $i \in \{\min(2 + u - s - |\alpha|, u - s), \dots, \max(0, u - s - |\alpha|)\}$, descending do
23:         for $\gamma \in \{0,1\}^i$ do ▷ Implemented with a single linear scan to only consider $\gamma$ with either $K_0$ or
            $K_1$ nonempty
24:             ▷ The sets $K_0$,$K_1$ are located contiguously in $S$ and are next to each other  ◁
25:             $K_0 \leftarrow$ elements of $S$ ending in $\gamma.(0)$
26:             $K_1 \leftarrow$ elements of $S$ ending in $\gamma.(1)$
27:             if first then
28:                 Sort $K_0$ by key $x \mapsto (\alpha x) \bmod 2^{\min(u, i+|\alpha|)}$
29:                 Sort $K_1$ by key $x \mapsto (\alpha x) \bmod 2^{\min(u, i+|\alpha|)}$
30:             else
31:                 FILTERMERGESTEP(&mut$K_0$, $i$)      ▷ Update sorting of the sets for the new, smaller i
32:                 FILTERMERGESTEP(&mut$K_1$, $i$)
33:             $\rho[i] \leftarrow \rho[i] + $ BATCHPAIRESTIMATE($K_0, K_1, \alpha, i$)   ▷ From Algorithm 2
34:         first ← false
35:     return $\rho$
```

6 Faster selection of OMA hash functions

OMA hash functions can be selected in a similar fashion to Section 5; the key differences are a slightly different collision estimator and a new, $O(nu)$-time phase to select the additive part of the function.

6.1 Multiplier selection

Let S be the given set of n values in $\{0,1\}^u$ which we wish to hash to $\{0,1\}^s$ with a function from $\mathcal{H}^{\mathrm{OMA}}$. Define, as in Section 5, for each $i \in [0,u)$, $\gamma \in \{0,1\}^i$, and $t \in \{0,1\}$, the set $S_{i,\gamma,t} := \{x \in S : x \operatorname{sel}[0,i+1) = \gamma.(t)\}$. Define

$$\phi(\alpha,i,\gamma) := \sum_{x\in S_{i,\gamma,0}} \sum_{y\in S_{i,\gamma,1}} \mathbb{E}[\Pr[h_{a,b}(x) = h_{a,b}(y)] | h_{a,b} \in \mathcal{H}^{\mathrm{OMA}}_{\alpha,\star}]$$

We want to select α, bit by bit, to keep the following expected collision count (Definition 1.5) less than or equal to its expectation:

$$\mathbb{E}[\kappa(h_{a,b},S)|h_{a,b} \in \mathcal{H}^{\mathrm{OMA}}_{\alpha,\star}] = \sum_{i\in[u)} \sum_{\gamma\in\{0,1\}^i} \phi(\alpha,i,\gamma)$$

For a given partial multiplier prefix $\alpha \in \{0,1\}^\star$, the way $\phi(\alpha,i,\gamma)$ can be evaluated depends chiefly on the value of i:

trivial region If $i \geq u-s$, then $\phi(\alpha,i,\gamma) = 0$.

base region If $i + |\alpha| \leq u-s$, then $\phi(\alpha,i,\gamma) = |S_{i,\gamma,0}||S_{i,\gamma,1}|/2^s$.

interactive region If $i \leq u-s$ and $i+|\alpha| > u-s$, then, with $t = \min(|\alpha|+i, u)$, by Lemma 3.3

$$\phi(\alpha,i,\gamma) = \frac{1}{2^{2u-s-t}} \sum_{x\in S_{i,\gamma,0}} \sum_{y\in S_{i,\gamma,0}} \max(0, 2^{u-s} - \operatorname{mabs}(\alpha x - \alpha y, 2^t))$$

If $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$ are provided as lists sorted by the key function $z \mapsto (\alpha z) \bmod 2^t$, then we can compute $\phi(\alpha,i,\gamma)$ using a similar linear-time scanning algorithm as Algorithm 2. [**explain: three indices, more casework heavy updates to maintain quantities. We do not present this because it takes a page.**] [**Doable with a similar rolling scan as the OMS case, except that three indices are needed, and to compute the values we maintain, in the loop, the sum of distances from the left endpoint. Total endpoint motion is limited, as before. This is casework intensive and we do not present the algorithm.**] !!! !!!

Most of the muliplier selection procedure for OMS hash functions can be reused here; the main difference is that Algorithm 2 will need to be modified to handle the more complicated objective function of Lemma 3.3.

6.2 Offset selection

Once a good multiplier a has been found, we choose the offset b by the method of conditional expectations; bits of b are chosen from in order from most significant to least significant. The quantity we maintain at less than or equal to its expectation is, for chosen high bits $\beta \in \{0,1\}^\star$:

$$\sigma(\beta) := \mathbb{E}[\kappa(h_{a,b}|h_{a,b} \in \mathcal{H}^{\mathrm{OMA}}_{a,\beta})] = \sum_{\{x,y\}\in\binom{S}{2}} \Pr[h_{a,b}(x) = h_{a,b}(y) \mid h_{a,b} \in \mathcal{H}^{\mathrm{OMA}}_{a,\beta})]$$

Algorithm 5 Efficient evaluation of $\sigma(\beta)$

```
▷ Requires: $S$ to be sorted by key $x \mapsto (ax) \bmod 2^u$ ◁
function ExpectedPartialOffsetCollisions($S$, $a$, $\beta$)
    $r_{\text{block}} \leftarrow -1$ ▷ Will be updated to be rightmost element with $aS[r_{span}] \bmod 2^u \operatorname{div} 2^{u-s}$ matching $aS[i] \bmod 2^u \operatorname{div} 2^{u-s}$, without wrapping around inside the region
    $r_{\text{span}} \leftarrow -1$ ▷ Will be updated to be rightmost element for which $(aS[r_{span}] - aS[i]) \bmod 2^{u-s} < 2^{u-s}$
    $o \leftarrow \beta 2^{u-s-|\beta|}$
    $g \leftarrow 2^{u-s} - 2^{u-s-|\beta|}$
    $p \leftarrow 0$
    $\texttt{acc} \leftarrow max(0, (aS[0]+o) \bmod 2^{u-s} - g)$ ▷ An accumulator over indices in $\{i+1, \dots, r_{span}\}$
    for $i$ in $0, \dots, |S|-1$ do
        $t \leftarrow max(0, (aS[i]+o) \bmod 2^{u-s} - g)$
        $\texttt{acc} \leftarrow \texttt{acc} - t$
        ▷ Advance $r_{span}$, without wrapping past $i$ ◁
        if $r_{\text{span}} = i-1$ then
            $r_{\text{span}} \leftarrow i$
        while $(r_{\text{span}}+1) \bmod |S| \neq i$ and $(aS[(r_{\text{span}}+1) \bmod |S|] - aS[i]) \bmod 2^{u-s} < 2^{u-s}$ do
            $r_{\text{span}} \leftarrow (r_{\text{span}}+1) \bmod |S|$
            $\texttt{acc} \leftarrow \texttt{acc} + \max(0, (aS[\text{span}]+o) \bmod 2^{u-s} - g)$
        ▷ Advance $r_{block}$, without wrapping past $i$'s block ◁
        if $r_{\text{block}} = i-1$ then
            $r_{\text{block}} \leftarrow i$
        while $(aS[(r_{\text{block}}+1) \bmod |S|]+o) \bmod 2^u \operatorname{div} 2^{u-s} = (aS[i]+o) \bmod 2^u \operatorname{div} 2^{u-s}$ and $(aS[(r_{\text{block}}+1) \bmod |S|]+o) \bmod 2^{u-s} > (aS[i]+o) \bmod 2^{u-s}$ do
            $r_{\text{block}} \leftarrow (r_{\text{block}}+1) \bmod |S|$
        $p \leftarrow p + ((r_{\text{block}}-i) \bmod |S|)2^{u-s-|\beta|} + t((r_{\text{span}}-i) \bmod |S|) - \texttt{acc}$
    return $p/2^{u-s-|\beta|}$
```

Assuming the input set S has already been sorted by key $x \mapsto ax \bmod 2^u$ beforehand, the quantity $\sigma(\beta)$ for any β can be computed in linear time, since by Lemma 3.4 the individual terms $\Pr[h_{a,b}(x) = h_{a,b}(y) \mid h_{a,b} \in \mathcal{H}^{\text{OMA}}_{a,\beta})]$ have a relatively simple formula. See Algorithm 5.

[**Should explain how algorithm 5 works – a single rolling scan captures and aggregates all interactions. Algorithm needs direct implementation from pseudocode to check typos.**] !!!

7 Faster selection of OCM hash functions

The selection of OCM hash functions is slightly simpler than for OMS and OMA hash functions. Again, we use the method of conditional expectations, selecting bits of the multiplier from least to most significant, using the decomposition into batches of pairs as described in Section 5 to efficiently evaluate the conditional expectation of the number of collisions within the input set.

Let S be the given set of n values in $\{0,1\}^u$ which we wish to hash to $\{0,1\}^s$ with a function from $\mathcal{H}^{\text{OCM}}$. Define, as in Section 5, for each $i \in [0,u)$, $\gamma \in \{0,1\}^i$, and $t \in \{0,1\}$, the set $S_{i,\gamma,t} := \{x \in S : x \operatorname{sel}[0,i+1) = \gamma.(t)\}$. Define

$$\phi(\alpha,i,\gamma) := \sum_{x \in S_{i,\gamma,0}} \sum_{y \in S_{i,\gamma,1}} \mathbb{E}[\Pr[h_a(x) = h_a(y)] | a \text{ extends } \alpha]$$

For a given partial multiplier prefix $\alpha \in \{0,1\}^\star$, there are three main regions for pairs (x,y) of elements as a function of $i := \textsc{tzcnt}(x \oplus y)$.

trivial region If $i \geq u - s$, then $\phi(\alpha, i, \gamma) = 0$.

base region If $i + |\alpha| \leq u - s$, then $\phi(\alpha, i, \gamma) = |S_{i,\gamma,0}||S_{i,\gamma,1}|/2^s$.

interactive region If $i \leq u - s$ and $i + |\alpha| > u - s$, then $\phi(\alpha, i, \gamma)$ is just $1/2^{u-|\alpha|-s}$ times the number of pairs (x,y) for $x \in S_{i,\gamma,0}$ and $y \in S_{i,\gamma,1}$ where $(\alpha \odot (x \oplus y)) \operatorname{sel}[u-s, i+|\alpha|) \neq 0$. If $S_{i,\gamma,0}$ and $S_{i,\gamma,1}$ are provided sorted by key function $x \mapsto \textsc{brev}((\alpha \odot x) \operatorname{sel}[u-s,u))$, then this number can be computed in linear time. (See Algorithm 6.)

Algorithm 6 Efficient evaluation of $\phi(\alpha, i, \gamma)$ in the interactive region, for the $\mathcal{H}^{\text{OCM}}$ hash family

```
▷ Requires: $S_0$ and $S_1$ to be sorted by key $x \mapsto \textsc{brev}((\alpha \odot x) \operatorname{sel}[u-s,u))$ ◁
function BatchPairEstimate($S_0$, $S_1$)
    $p \leftarrow 0$
    $(i_0, i_1) \leftarrow (0,0)$
    ▷ Iterate over maximum subsets $T_0 \subseteq S_0$ and $T_1 \subseteq S_1$ so that $T_0$ and $T_1$ share bits $[u-s, |\alpha|+i)$,
      and sum terms $|T_0||T_1|$ ◁
    while $i_0 < |S_0|$ and $i_1 < |S_1|$ do
        $(j_0, j_1) \leftarrow (i_0, i_1)$
        if $\textsc{brev}(S_0[i_0] \operatorname{sel}[u-s,u)) \leq \textsc{brev}(S_1[i_1] \operatorname{sel}[u-s,u))$ then
            while $j_0 < |S_0|$ and $S_0[j_0] \operatorname{sel}[u-s,|\alpha|+i) = S_0[i_0] \operatorname{sel}[u-s,|\alpha|+i)$ do
                $j_0 \leftarrow j_0 + 1$
        if $\textsc{brev}(S_1[i_1] \operatorname{sel}[u-s,u)) \leq \textsc{brev}(S_0[i_0] \operatorname{sel}[u-s,u))$ then
            while $j_1 < |S_1|$ and $S_1[j_1] \operatorname{sel}[u-s,|\alpha|+i) = S_1[i_1] \operatorname{sel}[u-s,|\alpha|+i)$ do
                $j_1 \leftarrow j_1 + 1$
        $p \leftarrow p + (j_0 - i_0)(j_1 - i_1)$
        $(i_0, i_1) \leftarrow (j_0, j_1)$
    return $p \cdot \frac{1}{2^{u-|\alpha|-i}}$
```

Sorted access to the $S_{i,\gamma,t}$ sets, as needed, is slightly simpler to maintain than it was in Section 5. [**Explain the modified sorting algorithm: modify the OMS approach, with the main difference being that FilterMergeStep *only* needs to merges the two sublists. And footnote: if randomization is acceptable, can skip the sorting and use a hash table to count common bit patterns in** $\operatorname{sel}[u-s,|\alpha|+i)$.] !!!

8 Constructing a minimal perfect hash function

Observation 8.1. *The OMS, OMA, and OCM hash families all have a* bijection-shift *structure; the functions in them apply a bijection* $\{0,1\}^u \mapsto \{0,1\}^u$ *to the input and then select the top s bits of the result.*

That the hash families have this structure follows from their linearity and use of an odd multiplier. Specifically, for any distinct pair $x,y \in \{0,1\}^u$*, and OMS hash function from* $\{0,1\}^u$ *to* $\{0,1\}^u$*,* $h_a(x) := ax \bmod 2^u$*,* $h_a(x) = h_a(y)$ *iff* $a(x-y) \bmod 2^u = 0$ *which only occurs if* $x = y$*, because a is odd. Similarly, for an OMA hash function* $h_{a,b}(x) := (ax+b) \bmod 2^u$*,* $h_{a,b}(x) = h_{a,b}(y)$ *iff* $a(x-y) \bmod 2^u = 0$*. Finally, for an OCM hash function* $h_a(x) := (a \odot x) \bmod 2^u$*,* $h_a(x) = h_a(y)$ *iff* $a \odot (x-y) \bmod 2^u = 0$ *which holds only iff* $x = y$*, because a is odd and multiplying by it preserves the least significant nonzero bit of* $x - y$*.*

Lemma 8.2 (Tighter analysis of [HMP01]'s displacement finding algorithm)**.** *Let r,t be nonnegative integers. For $i \in [n]$, let $f_i \in \{0,1\}^t$ and $g_i \in \{0,1\}^r$ be integers, so that all (f_i,g_i) pairs are distinct. For each $x \in \{0,1\}^t$, define $G_x := \{g_i : i \in [n] \wedge f_i = x\}$. Let $c = \sum_{x\in\{0,1\}^t} \binom{|G_x|}{2}$. If $\max(n,4c) \leq 2^r$, then running Algorithm 7 on $L = ((f_i,g_i))_{i\in[n]}$ will produce a displacement table $d : \{0,1\}^t \to \{0,1\}^r$ so that, for all $i \in [n]$, the value $g_i \oplus d(f_i)$ is unique.*

Algorithm 7 Deterministic displacement assignment from [HMP01, Section 4], with minor modifications

```
1: function FINDDISPLACEMENT($L$, $t$, $r$)        ▷ For $L$ a list of $(\{0,1\}^t, \{0,1\}^r)$ pairs, all distinct
2:     Form list $K$ of (subset, key) pairs from $L$
3:     Sort $K$ in decreasing order of subset size, handling ties arbitrarily
4:     $D \leftarrow \vec{0} \in (\{0,1\}^r)^{2^t}$
5:     for $i$ in $0,\dots,r$ do
6:         $M_i \leftarrow \vec{0} \in \mathbb{N}^{2^{i+1}}$
7:     for $(f,G)$ in $K$ do
8:         $d \leftarrow 0 \in \{0,1\}^r$
9:         $w \leftarrow |G| M_0[0]$
10:        for $i$ in $1,\dots,r$ do
11:            $w_0 \leftarrow 0$
12:            for $g$ in $G$ do
13:                $w_0 \leftarrow w_0 + M_i[(g \operatorname{div} 2^{r-i}) \oplus (2d)]$
14:            $w_1 \leftarrow w - w_0$
15:            if $w_1 < w_0$ then
16:                $d \leftarrow (2d) \oplus 1$
17:                $w \leftarrow w_1$
18:            else
19:                $d \leftarrow 2d$
20:                $w \leftarrow w_0$
21:        for $i$ in $0,\dots,r$ do
22:            for $g$ in $G$ do
23:                $M_i[(g \oplus d) \operatorname{div} 2^{r-i}] \leftarrow M_i[(g \oplus d) \operatorname{div} 2^{r-i}] + 1$
24:        $D[f] \leftarrow d$
25:    return $D$
```

Proof. Let $F := \{f_i : i \in [n]\}$, and let $x_1,\dots,x_{|F|}$ be the elements in F ordered by descreasing value of $|G_x|$, so that $|G_{x_1}| \geq |G_{x_2}| \geq \dots \geq |G_{x_{|F|}}|$. A described in [HMP01, Section 4.3], Lemma 8.2 iterates over $x_1,\dots,x_|F|$ and, using the method of conditional expectations, assigns a displacement value $d(x_i)$ to each x_i so that $|G_{x_i}|$ has no more than the expected number of intersections, if $d(x_i)$ were chosen uniformly at random from $\{0,1\}^r$, with $d(x_1) \oplus G_{x_1},\dots,d(x_{i-1}) \oplus G_{x_{i-1}}$, counting multiplicity. Consequently, there will be zero intersections if

$$v_i := |G_{x_i}| \sum_{j=1}^{i-1} |G_{x_j}| < 2^r \,.$$

Since $\sum_{j=1}^{|F|} |G_{x,j}| = n$, $v_i \leq |G_{x_i}|(n - |G_{x_i}|) < n|G_{x_i}|$. Also, if $|G_{x_j}| \geq 2$, then

$$v_i \leq \sum_{j=1}^{i-1} |G_{x_j}|^2 \leq \frac{2|G_{x,i}|}{|G_{x,i}|-1} \sum_{j=1}^{i-1} \binom{|G_{x_j}|}{2} < 4c$$

Thus $v_i < \max(n, 4c)$, so it $\max(n, 4c) \leq 2^r$ the algorithm will uniquely place all $d(f_i) \oplus g_i$. □

While Lemma 8.2 is a relatively tight analysis of Algorithm 7, it has not been ruled out whether there are other packing algorithms with stronger bounds.

Using the structure of the bijection-shift hash families (Observation 8.1), we can identify a bijection of values in $[O(n^2))$ so that the expected number of collisions when we extract the high bits is small. Then we can apply [HMP01]'s displacement table finding algorithm to construct a perfect hash function; but as the following theorem, shows, by processing the colliding and non-colliding values separately, we can obtain a *minimal* perfect hash function requiring a single dependent read of $\lceil \log n \rceil$ bits.

The general trick here to obtain a single-dependent-read evaluation, is tofirst find displacement values for values that collide under h and then place the keys that were already uniquely hashed by h into the gaps.

Theorem 8.3. *For n, u positive integers with $u \geq \log n$, implementing Algorithm 8 using either OMA or OCM hashes will construct a minimal perfect hash function f for a n-element subset S in $\{0,1\}^u$ to $[n]$, using $O(nu \log n)$ time, $O(nu)$ auxiliary space; the hash function f itself uses $\leq 5.66n\lceil \log n \rceil + O(u + \log n)$ bits of space and can be evaluated with exactly one dependent read of a $\lceil \log n \rceil$-bit word; there are $O(u + \log n)$ bits of parameters that are unconditionally used). OMS hashes can also be used but double the bound on the space usage.*

Algorithm 8 Minimal perfect hash function construction and evaluation

```
▷ $S$ is a set of $n$ distinct integers ◁
▷ $\mathcal{H}$ a class of bijection-shift type hash functions which is $\upsilon$-approximately universal ◁
function CONSTRUCT($S$, $\mathcal{H}$, $\upsilon$)
    $s \leftarrow \lceil \log(\upsilon\binom{n}{2} + 1) \rceil$
    $r \leftarrow \lfloor \log n \rfloor$
    $t \leftarrow \min(s, \lceil \log(4\upsilon\binom{n}{2}) \rceil - r)$        ▷ If $t < s$, this ensures $4\upsilon\binom{n}{2}/2^t \leq 2^r$
    Select $f : \{0,1\}^u \to \{0,1\}^s$ from $\mathcal{H}$ which is perfect on $S$
    $R \leftarrow \{f(x) : x \in S\}$.
    Select $g : \{0,1\}^s \to \{0,1\}^s$ from $\mathcal{H}$ so that $h(x) := g(x) \operatorname{div} 2^{s-t}$ has no more than $\upsilon\binom{n}{2}/2^t$ collisions on $R$
    Split $R$ into $R_H := \{x \in R : |h^{-1}(h(x)) \cap R| > 1\}$ and $R_L := \{x \in R : |h^{-1}(h(x)) \cap R| = 1\}$.
    ▷ Displace keys in $R_H$ to $[2^{\lfloor \log n \rfloor})$ ◁
    $L \leftarrow \{(h(x), g(x) \mod 2^{s-t}) : x \in R_H\}$
    $d \leftarrow$ FINDDISPLACEMENT$(L, t, r)$        ▷ This uses $s - t \leq r$
    ▷ Displace keys in $R_L$ to remaining locations in $[n)$ ◁
    $M \leftarrow \vec{0} \in \mathbb{N}^n$
    for $x \in R_H$ do
        $M[d[h(x)] \oplus (g(x) \mod 2^{s-t})] \leftarrow 1$
    for $x \in R_L$ do
        $w \leftarrow$ index of first zero entry in $M$
        $M[w] \leftarrow 1$
        $d[h(x)] \leftarrow w \oplus (g(x) \mod 2^{s-t})$        ▷ Now $d$ has $\lceil \log n \rceil$-bit entries
    return $((s, t, r, f, g, d))$

function EVAL($(s, t, r, f, g, d)$, $x$)
    $y \leftarrow g(f(x))$
    return $d[y \operatorname{div} 2^{s-t}] \oplus (y \mod 2^{s-t})$
```

Proof. The general construction, described in Algorithm 8, works with any class of bijection-shift hash family which from which functions can be selected with collision counts below the υ-universal expected value, for some υ. [**a bit awkward phrasing to handle OMS's selection method**] It requires two hash functions; one from $\{0,1\}^u$ to $\{0,1\}^s$ for some s, and one from $\{0,1\}^s$ to $\{0,1\}^s$ (which is chosen to reduce collisions for the top t bits of the output.) !!!

If the OMS, OMA, or OCM hash families are used, then the functions f and g can be selected using Theorem 1.6, Theorem 1.7, or Theorem 1.8 respectively. All of these have the same asymptotic time and space requirements. To select g, we apply the theorem to get a $\{0,1\}^s$ to $\{0,1\}^t$ hash function h but then skip the final $\operatorname{div} 2^{s-t}$ operation. Then $h(x) = g(x) \operatorname{div} 2^{s-t}$, and because g is a bijection, the pairs $((h(x), g(x) \bmod 2^{s-t}))$ are all unique. This is one of the prerequisites to apply Lemma 8.2 to the list L.

Another requirement is that the input pair elements are in $\{0,1\}^t$ and $\{0,1\}^r$. The former is immediate; the latter holds because $4\upsilon\binom{n}{2} \geq \upsilon\binom{n}{2} + 1$, which implies that

$$s - t = \max(0, \left\lceil \log(\upsilon\binom{n}{2} + 1) \right\rceil - (\left\lceil \log(4\upsilon\binom{n}{2}) - r \right\rceil)) \leq r$$

so for all $x \in R_H$, $g(x) \bmod 2^{s-t} \in [2^{s-t}) \subseteq [2^r)$.

The last condition to have Lemma 8.2 work is that $\max(|R_h|, 4\kappa(h, R_h))$ should be less than 2^r. First, since R_h contains only values x with $|h^{-1}(h(x)) \cap R| \geq 2$ and $\max_{z \geq 2} z/\binom{z}{=}2$, $|R_h| \leq 2\kappa(h, R_h)$. Note that

$$4\kappa(h, R_h) = 4\kappa(h, R) := 4\upsilon\binom{n}{2}/2^t$$

is $\leq 2^r$ if and only if

$$t \geq \left\lceil \log(4\upsilon\binom{n}{2}/2^r) \right\rceil = \lceil \log(2\upsilon n(n-1) \rceil - \lfloor \log n \rfloor$$

which holds by the definition of t.

[**lemmatize the $k(x)$ bound and move to appendix**] We now determine a bound on size of the the displacement table d. For any real a, the function $k : x \mapsto \lceil a + 2x - \lfloor x \rfloor \rceil - x$ is decreasing except at its possible discontinuities: when $a + 2x \bmod 1 = 0$ (where it may increase) and when $x \bmod 1 = 0$ (where it may decrease). It is also invariant under integral translations of x. We have two cases, depending on $a \bmod 1$. !!!

- If $a \bmod 1 = 0$, then the discontinuities in $[0,1)$ are at 0 and $1/2$; at 0, $k(x) = \lceil a \rceil$, while both one-sided limits are $\lceil a \rceil + 1$. At $1/2$, $k(x) = \lceil a \rceil + 1/2$, equaling the limit from the left, while the limit from the right is $\lceil a \rceil + 3/2$.
- If $a - \lfloor a \rfloor = a \bmod 1 \neq 0$, then the discontinuities in $[0,1)$ are at 0 (decreasing), $1/2 - \frac{a - \lfloor a \rfloor}{2}$ (increasing), and $\frac{a - \lfloor a \rfloor}{2}$ (increasing). The limit from the right at $1/2 - \frac{a - \lfloor a \rfloor}{2}$ is $\lceil a \rceil + \frac{1}{2} + \frac{a - \lfloor a \rfloor}{2}$, and the limit from the right at $1 - \frac{a - \lfloor a \rfloor}{2}$ is $\lceil a \rceil + 1 + \frac{a - \lfloor a \rfloor}{2}$. In both cases, the limit from the right is $\leq \lceil a \rceil + 3/2$.

Therefore, $\sup_{x \in \mathbb{R}} k(x) \leq 3/2$. With this we can bound

$$t = \lceil \log(2\upsilon) + 2\log n - \lfloor \log n \rfloor \rceil \leq \lceil \log(2\upsilon) \rceil + \frac{3}{2} + \log n\,.$$

(When υ is a power of two, and n is also exactly a power of two, we have that $\lfloor \log n \rfloor$ equals $\log n$, and $\lceil \log n(n-1) \rceil)$ equals $2\log n$, in which case $t = \log(2\upsilon) + \log n$. Decreasing n by one would increase t (because r is now smaller), and increasing n by one would increase t (to keep the collisions below the threshold).)

In particular, for the OMA family, where $\upsilon = 2$, the table always has $2^t \leq 2^{7/2} n \leq 11.31n$ entries. When using OMS or OCM hashes, where $\upsilon = 1$, the table always has $2^t \leq 2^{5/2} n \leq 5.66n$ entries. □

Remark. If one does not need a minimal perfect hash, then changing the parameters of Algorithm 8 to $r = \lceil \log n \rceil$ will slightly reduce the size of the displacement table.

Remark. For comparison, a tighter analysis of the double-displacement hashing scheme of [HMP01] suggests that for integer n and $r = \left\lceil \log \sqrt{2}n \right\rceil \leq \log(n) + \frac{3}{2}$, it can perfectly hash n element sets of elements in $\{0,1\}^{2r}$ to $\{0,1\}^r$ using two displacement tables with 2^r elements. The total number of displacement table entries would then be $\leq 2(2^{3/2}n)$, which is the same bound as Theorem 8.3's $2^{5/2}n$ when using OMA or OCM hashes.

Remark. Algorithm 8 can be modified to use the leaf-hashing construction of [FKS84] instead of the XOR-displacement procedure (Algorithm 7). If the leaf hash families are chosen to have $O(\ell^2)$ selection for leaf size ℓ, then the overall leaf hashing step will take $O(n)$ time. Depending on the ratios of the actual leaf hash ranges to $\binom{\ell}{2}$, the table of leaf hashes and offsets may end up with a smaller number of entries than Algorithm 8; however, these entries require a $\lceil \log n \rceil$ bit offset, plus for leaf size ℓ, an $\Omega(\log \ell + \log\log n)$[6] bit hash function

9 Open problems

Question 9.1. *Is there any near-linear time deterministic construction of a perfect hash function from* $\left[O(n^2)\right)$ *to* $[O(n))$ *using* $O(n)$ *bits of space,* and *which requires at most a single "dependent read" of a* $O(\log n)$ *bit interval to evaluate?*

Question 9.2. *Many perfect hash tables use variable length strings as keys, instead of fixed length integers. [Ruz09, Sections 5-6] describes a recursive method to deterministically find non-colliding hash signatures for strings, but the resulting hash function is not as simple as those of randomized hash function constructions.*

Is there any efficient deterministic parameter selection algorithm for the standard rolling polynomial string hash? Specifically, say $x^{(1)},\ldots,x^{(n)}$ *are strings in* $(\{0,1\}^k)^\star$*, and* $\mathbb{F}$ *is a finite field of size* $\leq 2^{O(k)}$*. (For example, a Mersenne prime field like* $\mathbb{Z}/(2^{89}-1)\mathbb{Z}$*.) For* a *in* $\mathbb{F}$*, define the hash function* $h_a(x) = \sum_{i=0}^{|x|-1} a^i x_i$*. If* a *is chosen uniformly at random, then the probability that all* $h_a(x^{(j)})$ *for* $j \in [n]$ *are distinct is*

$$\geq 1 - \frac{1}{|\mathbb{F}|} \sum_{\{i,j\}\in\binom{[n]}{2}} \max(|x^{(i)}|,|x^{(j)}|) \geq 1 - \frac{1}{|\mathbb{F}|} n \sum_{i\in[n]} |x^{(i)}|$$

For input size $t = n\sum_{i\in[n]} |x^{(i)}|$*, is there any* $O(t\,\mathrm{polylog}(t))$ *time deterministic algorithm to find a value of* a *without any collisions?* $O(tn\,\mathrm{polylog}(t))$ *is certainly doable using multipoint polynomial evaluation (Lemma B.1).*

Question 9.3. *Can any of the OMS, OMA, or OCM hash selection algorithms be optimized to use* $O(n(u + \log n))$ *asymptotic (word-RAM) time, or implemented to* always *run within a small constant factor of sorting time* in practice*? (To give a specific target, below* 100 *ns per element when processing a set of* $\leq 2^{25}$ *values in* $\{0,1\}^{64}$*; unoptimized implementations of the current algorithms take around* 10000 *ns per element?)*

[6]This is a well known lower bound, with the following short proof. Say $\mathcal{H}$ is a universal hash family from A to B; every pair $x,y \in A$ must be distinguished by some $h \in \mathcal{H}$, so the function $f : A \to B^{|\mathcal{H}|}$ which concatenates all $h \in \mathcal{H}$ must be injective, hence $|\mathcal{H}| \geq \log|A|/\log|B|$; and the most common hash function in $\mathcal{H}$ will collide on some pair w,z of elements in A and so must be chosen with $\leq 1/|B|$ probability, so $|\mathcal{H}| \geq |B|$. Together these imply $\log|\mathcal{H}| \geq \Omega(\log\log|A| + \log|B|)$.

References

[AN96] Noga Alon and Moni Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4):434–449, 1996.

[BCF$^+$21] Michael A. Bender, Alex Conway, Martin Farach-Colton, William Kuszmaul, and Guido Tagliavini. All-purpose hashing. *CoRR*, abs/2109.04548, 2021.

[BES05] Alin Bostan and Éric Schost. Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, 21(4):420–446, 2005. Festschrift for the 70th Birthday of Arnold Schonhage.

[CHM92] Zbigniew J Czech, George Havas, and Bohdan S Majewski. An optimal algorithm for generating minimal perfect hash functions. *Information processing letters*, 43(5):257–264, 1992.

[CW79] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[DHKP97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.

[FK84] Michael L. Fredman and János Komlós. On the size of separating systems and families of perfect hash functions. *SIAM Journal on Algebraic Discrete Methods*, 5(1):61–68, 1984.

[FKS84] Michael L Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $o(1)$ worst case access time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984.

[FN93] Amos Fiat and Moni Naor. Implicit o(1) probe search. *SIAM Journal on Computing*, 22(1):1–10, 1993.

[HMP01] Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh. Deterministic dictionaries. *Journal of Algorithms*, 41(1):69–85, 2001.

[HvdH21] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n\log n)$. *Annals of Mathematics*, 193(2):563 – 617, 2021.

[Kra94] Hugo Krawczyk. Lfsr-based hashing and authentication. In *Annual International Cryptology Conference*, pages 129–139. Springer, 1994.

[Ram95] Rajeev Raman. Improved data structures for predecessor queries in integer sets. Technical report, Citeseer, 1995.

[Ruz08a] Milan Ruzic. Constructing efficient dictionaries in close to sorting time. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2008.

[Ruz08b] Milan Ruzic. Uniform deterministic dictionaries. *ACM Trans. Algorithms*, 4(1):1:1–1:23, 2008.

[Ruz09] Milan Ruzic. Making deterministic signatures quickly. *ACM Trans. Algorithms*, 5(3):26:1–26:26, 2009.

[Tho15] Mikkel Thorup. High speed hashing for integers and strings. *arXiv preprint arXiv:1504.06804*, 2015.

A Deferred proofs

B Deterministic selection of polynomial string hash parameters

Lemma B.1 (Deterministic selection of string hashing parameters)**.** *Let $\mathbb{F}$ be a finite field and let $x^{(1)},\ldots,x^{(n)}$ be unique strings in $\mathbb{F}^\star$. Let $t = n\sum_{i\in[n]}|x^{(i)}|$ be the number of input characters. For each $a \in \mathbb{F}$, define $h_a(x) := \sum_{i=0}^{|x|-1} a^i x_i$. If $t < |\mathbb{F}|$, then there is a deterministic algorithm taking $O(tn\log|\mathbb{F}|\log(tn\log\mathbb{F})\log(tn))$ finite-tape-Turing-machine time to identify a specific $a_\star \in \mathbb{F}$ for which $h_{a_\star}(x^{(1)}),\ldots,h_{a_\star}(x^{(n)})$ are all distinct.*

Proof. Equivalently, we seek to find a value a for which, for all $\{i,j\} \in \binom{[n]}{2}$, $h_a(x^{(i)}) \neq h_a(x^{(i)})$. For a given i,j, this occurs if the polynomial

$$P_{i,j}(a) := \sum_{k=0}^{\max(|x^{(i)}|,|x^{(j)}|)-1} a(x_k^{(i)} - x_k^{(j)})$$

does *not* have a root at a. It thus suffices to find a value which is not a root of

$$Q(a) := \prod_{\{i,j\}\in\binom{[n]}{2}} P_{i,j}(a)\,.$$

The polynomial Q has degree

$$\sum_{\{i,j\}\in\binom{[n]}{2}} (\max(|x^{(i)}|,|x^{(j)}|)-1) < n\sum_{i\in[n]}|x^{(i)}| =: t\,.$$

It can be constructed using a tree of products that starts by multiplying individual pairs of $P_{i,j}$ polynomials and culminates in the multiplication of two degree-$O(t)$ polynomials. Since Q has at most $\deg Q$ roots in $\mathbb{F}$, if we evaluate Q on an arbitrary set S of t elements in $\mathbb{F}$, there will be at least one $a_\star \in S$ for which $Q(a_\star) \neq 0$. Since $Q(a_\star) \neq 0$, it follows that for all $\{i,j\} \in \binom{[n]}{2}$, $h_{a_\star}(x^{(i)}) \neq h_{a_\star}(x^{(i)})$, hence $a_\star$ is the desired value.

Define $M(s)$ to be the Turing-machine time needed to multiply two degree s polynomials over $\mathbb{F}$, and $E(s)$ the time to evaluate a degree s polynomial at s points, then the total runtime of this procedure is $O(M(tn)\log(tn) + E(tn))$. According to [BES05], $E(s) = O(3/2M(s)\log s + O(M(s)))$ (although if S is a geometric progression one can get $E(s) = O(M(s))$). Since $s = O(\text{poly}\,|\mathbb{F}|)$, we can use Kronecker substitution to multiply degree s polynomials using integer multiplication of $O(s\log|\mathbb{F}|)$ bit quantities, which by [HvdH21] takes $O(s\log|\mathbb{F}|\log(s\log\mathbb{F}))$ time on a multi-tape Turing machine. Combining these results, the total time required to find $a_\star$ is $O(tn\log|\mathbb{F}|\log(tn\log\mathbb{F})\log tn)$. $\square$

C Runtime evaluation for previous work

[**Ruzic's universe reduction: what is the Turing machine bit complexity? What changes with the improved inversion count approximation algorithm of Chan and Patrascu?**] !!!