

6.856 Project: Bounds on the Independence Required for Cuckoo Hashing

Jeffrey Cohen and Daniel M. Kane

December 31, 2005

Abstract

In Cuckoo Hashing, we achieve expected $O(1)$ amortized time per operation for dynamic hashing by using two hash functions and maintaining the property that each key is hashed to the value indicated by either of the hash functions [1]. This result is known as long as the hash functions are chosen independently and have $O(\lg n)$ -independence. In this paper, we show that only one of the hash functions must exhibit $O(\lg n)$ -independence, and the other only needs to be 2-independent. We also show that 5-independence or less for both hash functions is insufficient. Furthermore, if the hash functions are not chosen independently of one another but still satisfy a condition that we call *joint-independence*, $\Omega(\lg n)$ -joint-independence is both necessary and sufficient to guarantee that Cuckoo Hashing will work.

1 Introduction

In Cuckoo Hashing, we achieve expected $O(1)$ amortized time per operation for dynamic hashing by using two hash functions and maintaining the property that each key is hashed to the value indicated by either of the hash functions [1]. This result is known as long as the hash functions are chosen independently and have $O(\lg n)$ -independence. In this paper, we show that only one of the hash functions must exhibit $O(\lg n)$ -independence, and the other only needs to be 2-independent. We also show that 5-independence or less for both hash functions is insufficient. Furthermore, if the hash functions are not chosen independently of one another but still satisfy a condition that we call *joint-independence*, $\Omega(\lg n)$ -joint-independence is both necessary and sufficient to guarantee that Cuckoo Hashing will work.

In Section 2 of this paper, we give a description of different types of independence that can be exhibited by families of hash functions. In Section 3, we present a useful concept known as *collision space* that describes which keys result in collisions between two hash functions. We also discuss the meanings of types of independence based on representations of hash families in collision space. In Section 4 we present our results on Cuckoo Hashing.

2 Types of Independence

In this section we discuss definitions for the types of independence for families of pairs of hash functions that we will consider in this paper. Our aim is to obtain bounds on the degree of independence needed for successful Cuckoo Hashing.

A family of hash functions $h : U \rightarrow \{1, 2, \dots, m\}$ is said to be k -independent if, for any k distinct keys $x_1, \dots, x_k \in U$, and any values $a_1, \dots, a_k \in \{1, 2, \dots, m\}$, then for a random hash function in the family,

$$\Pr(h(x_i) = a_i \forall i) = \frac{O(1)}{m^k}.$$

A family of pairs of hash functions is (k_1, k_2) -independent if it is obtained by picking hash functions from k_1 - and k_2 -independent hash families independently of one another.

We define a family of pairs of hash functions h_1, h_2 as k -joint-independent if for any k distinct keys x_1, x_2, \dots, x_k and $2k$ values a_1, a_2, \dots, a_k and b_1, b_2, \dots, b_k , for a randomly chosen pair of hash functions,

$$\Pr(h_1(x_i) = a_i \text{ and } h_2(x_i) = b_i \forall i) = \frac{O(1)}{m^{2k}}.$$

Note that having (k, k) -independence is equivalent to having both k -joint-independence and independent selection of the hash functions.

3 Description of Collision Space

Collision space is used in our proofs of our bounds. Collision space for hash functions h_1 and h_2 is a graph whose vertices are the elements of U and with two edge sets, E_1 and E_2 .

$$E_i = \{(x, y) : h_i(x) = h_i(y)\}.$$

Definition. A hash family is called value-oblivious if for any permutation π mapping our values to themselves, the hash function f is chosen with the same probability as $\pi(f)$.

Note that if a hash family is value-oblivious, it is uniquely defined by the representation of its hash functions in collision space. In value-oblivious hash families, there is a simple characterization of independence in collision space.

Lemma 1. If $k = o(\sqrt{n})$, then a value-oblivious hash family is k -independent if and only if for any k distinct keys x_1, \dots, x_k and any equivalence relation, \sim , on the x_i that splits them into c equivalence classes

$$\Pr(h(x_i) = h(x_j) \forall x_i \sim x_j) = \frac{O(1)}{m^{k-c}}. \quad (1)$$

Proof. The equation above clearly holds for any k -independent hash family by the definition of k -independence. We will use the convention that for a function, f that $f((a, b, c, \dots)) = (f(a), f(b), f(c), \dots)$. For any keys x_1, \dots, x_k and values a_1, \dots, a_k , let \sim be the equivalence relation $x_i \sim x_j$ if $a_i = a_j$. Let c be the number of equivalence classes under \sim . Notice that over permutations π the range of $\pi((a_1, a_2, \dots, a_k))$ has order $m(m-1)\dots(m-c+1)$. By value-obliviousness, $\Pr(h((x_1, \dots, x_k)) = (a_1, \dots, a_k)) = \Pr(h((x_1, \dots, x_k)) = \pi((a_1, \dots, a_k)))$. Since the left hand side of equation (1) is less than the sum of this probability over all possible values of $\pi((a_1, \dots, a_k))$, we have that

$$\begin{aligned} \Pr(h((x_1, \dots, x_k)) = (a_1, \dots, a_k)) \\ \leq \frac{1}{m(m-1)\dots(m-c+1)} \Pr(h(x_i) = h(x_j) \forall x_i \sim x_j) \\ \leq \left(\frac{O(1)}{m^{k-c}}\right) \left(\frac{O(1)}{m^c}\right) = \frac{O(1)}{m^k}. \end{aligned}$$

□

Definition. A family of pairs of hash functions is called joint-value-oblivious if for any permutations π and ϕ mapping our values to themselves, the hash functions (f, g) are chosen with the same probability as $(\pi(f), \phi(g))$.

We have a similar collision space characterization of joint-independence for joint-value-oblivious families.

Lemma 2. If $k = o(\sqrt{n})$, then a joint-value-oblivious hash family is k -joint-independent if and only if for any k distinct keys x_1, \dots, x_k and any equivalence relations, \sim_l , on the x_i that split them into c_l equivalence classes (for $l = 1, 2$)

$$\Pr(h_l(x_i) = h_l(x_j) \forall x_i \sim_l x_j) = \frac{O(1)}{m^{2k-c_1-c_2}}. \quad (2)$$

Proof. The proof of this statement is analogous to that of Lemma (1). □

4 Cuckoo Hashing

4.1 Overview of Cuckoo Hashing

Cuckoo Hashing is a technique for solving dynamic dictionary problems in expected $O(1)$ amortized time per operation, first described by Pagh and Rodler [1]. In Cuckoo Hashing, we choose hash functions h_1 and h_2 from two hash families and guarantee that each key x in the input is hashed to either $h_1(x)$ or $h_2(x)$. If we get a collision during an insertion, we rearrange elements y , switching y from $h_1(y)$ to $h_2(y)$ (or vice versa) until we have no collisions. If we get a chain of replacements that is too long or forms a loop (in which case we cannot eliminate all collisions), we rehash.

Pagh and Rodler have shown that if the hash families are $O(\lg n)$ -independent, Cuckoo Hashing works in expected $O(1)$ amortized time per operation. We will

give additional results stating that it is sufficient for one hash function to be $O(\lg n)$ -independent, as well as lower bounds on the degree of independence needed to guarantee that Cuckoo Hashing will work.

4.2 Sufficiency of $(O(\lg n), 2)$ -Independence

Cuckoo hashing is known to work for an $(O(\lg n), O(\lg n))$ -independent hash family [1]. We now show that it works in $O(n)$ space and $O(1)$ expected amortized time for an $(O(\lg n), 2)$ -independent hash family with slight modification to Pagh and Rodler's algorithm [1]. To prove this result, we will consider the cases which are problematic for Cuckoo Hashing and show that these are unlikely. These cases are: (1) any cycle of length $O(\lg n)$ in the collision-space graph where edges alternate between E_1 and E_2 , (2) collision-space graphs where the sum over all paths of length $O(\lg n)$ with edges alternating between E_1 and E_2 of the lengths of those paths is $\Omega(n)$, and (3) any path of length $\Omega(\lg n)$ with edges alternating between E_1 and E_2 .

Case (1) is problematic for Cuckoo Hashing because it corresponds to potential loops in the insertion procedure as explained by Pagh and Rodler [1]. Case (2) is problematic because it corresponds to the potential for chains of displacements taking a total of $\Omega(n)$ time. Case (3) is problematic because it would allow for cycles and paths of length longer than our bound in cases (1) and (2). From this it is clear that if none of these cases is violated, Cuckoo Hashing runs with the appropriate time bounds.

We will show that, for $m \geq cn$, for an appropriate constant c , we avoid violating any of these cases with probability at least $\frac{1}{2}$. We will use the same algorithm as Pagh and Rodler with the exception of rehashing if we perform more than $c_t n$ operations for some constant c_t . We may need this modification because our analysis allows for bad hash functions that take excessive time to hash without leading to any of the problems in Pagh and Rodler's algorithm.

We can think of our hash functions as choosing the 2-independent one (h_2) first, then the $O(\lg n)$ -independent one ($h_{\lg n}$). Notice that for the 2-independent function, the expected number of pairs of keys that collide is $O(\frac{n^2}{m})$. If m is a sufficiently large constant times n , with probability at least $\frac{7}{8}$, the total number of collisions is at most $\frac{m}{C} - n$ for a large constant C . Call the equivalence classes for the hashed elements under h_2 (that is, the sets of keys hashing to the same value) L_1, L_2, \dots, L_k of sizes A_1, A_2, \dots, A_k . The sum of the squares of the A_i 's is n plus the number of collisions. With probability at least $\frac{7}{8}$, this is at most $\frac{m}{C}$. We will consider the probabilities of avoiding the three problematic cases when this is true.

Case 1 (cycles of length $O(\lg n)$): We want to consider the expected number of cycles whose vertices lie in equivalence classes in a pattern like

$$L_{i_1}, L_{i_1}, L_{i_2}, L_{i_2}, \dots, L_{i_l}, L_{i_l}.$$

(We repeat each index because every other edge appears in the set of edges representing collisions in h_2 . The edges going between equivalence classes represent

collisions in $h_{\lg n}$.) The number of such cycles is $\prod_{j=1}^l A_{i_j}^2$. This is obtained by counting the number of ways to pick two vertices (keys) from each of the appropriate equivalent classes. For alternating edges to be collisions of $h_{\lg n}$, we need l pairs of vertices to collide. Because $h_{\lg n}$ is more than $2l$ -independent, the probability that a particular cycle of length l is in our graph is $O\left(\frac{1}{m^l}\right)$. Thus the probability that any of these cycles is in our graph is at most $O\left(\prod_{j=1}^l \frac{A_{i_j}^2}{m}\right)$.

The probability that we have any cycle of length l is at most

$$O\left(\sum_{i_1, i_2, \dots, i_l} \prod_{j=1}^l \frac{A_{i_j}^2}{m}\right) = O\left(\left(\sum_{i=1}^k \frac{A_i^2}{m}\right)^l\right) = O\left(\frac{1}{C^l}\right).$$

Summing over all l less than $O(\lg n)$ if C is sufficiently large, the expected number of cycles of any length $O(\lg n)$ is at most $\frac{1}{8}$. Hence, with probability $\frac{7}{8}$, we have no cycles of these lengths.

Case 2 (path-lengths sum to $\Omega(n)$): From our above analysis, we see that the expected number of paths $L_{i_1}, L_{i_1}, L_{i_2}, L_{i_2}, \dots, L_{i_l}, L_{i_l}$ is $O\left(m \prod_{j=1}^l \frac{A_{i_j}^2}{m}\right)$.

We have the same collisions, and the additional factor of m comes from the fact that we no longer require a collision between the first and last vertices. By the analysis in case (1), the expected number of paths of length $2l - 1$ of this form is $O\left(\frac{m}{C^l}\right)$.

Note that any path of length l is contained in a path of length at most $l + 2$ that starts and ends with edges corresponding to collisions of h_2 (that is, a path of the form used above). Hence we only have to consider paths of this form. We know from the above that the expected number of such paths is $O\left(\frac{m}{C^{\frac{l}{2}}}\right)$. (The l in the exponent of the denominator is halved because we are looking at path length, rather than number of equivalence classes.) Hence summing over all paths, the expected value of the sum of path lengths is $O(m) = O(n)$, so for a sufficiently large constant D , the probability that this sum is actually more than Dn is at most $\frac{1}{8}$.

Case 3 (paths of length $\Omega(\lg n)$): For a particular l that is $\Omega(\lg n)$ and such that $2l$ is smaller than our degree of independence, the expected number of paths of length l is $\frac{O(m)}{C^{\frac{l}{2}}} \leq \frac{1}{8}$. Hence with probability $\frac{7}{8}$, there are no paths of this length, and thus no paths of longer length.

Hence with probability at least $\frac{7}{8}$, the sum of the A_i^2 's is less than $\frac{m}{C}$, and if this holds, we avoid each of our three problematic cases with probability at least $\frac{7}{8}$. Hence we avoid all of our problematic cases with probability at least $1 - \frac{4}{8} = \frac{1}{2}$.

We have now shown that our three problematic cases occur with probability less than $\frac{1}{2}$. Thus we need to rehash expected $O(1)$ times per n insertions, and each rehashing takes $O(n)$ time. Therefore the expected amortized time for n operations is $O(1)$.

4.3 Failure of 2- and 3-Independence

We prove that 2- and 3-independence are not sufficient for Cuckoo Hashing to work in expected $O(1)$ time per operation and linear space. In particular we construct 2- and 3-independent hash families that force Cuckoo Hashing to rehash with high probability. Although the result that 3-independence is insufficient implies that 2-independence is insufficient, we include proofs of both results because the 2-independence proof is simpler and instructive for understanding the 3-independence proof.

For simplified analysis, we will assume that our universe is of size n (we can assume this because it is sufficient for our hash functions to fail on only one set of inputs). We also note that we can change the value of n by some constant multiplicative factor without affecting the problem. Hence we can assume for example, that n is a power of k for some constant k .

Proposition 3. *If n is a power of 3, there exists a 2-independent family of hash functions, so that if two functions are chosen independently from the family, Cuckoo Hashing with these hash functions avoids rehashing with probability $\left(1 - \Omega\left(\frac{n}{m}\right)^5\right)^{n/9}$. Therefore, the expected amortized time to hash n items with these families is at least $\exp\left(\Omega\left(\frac{n^6}{m^5}\right)\right)$.*

Proof. We associate our keys with elements of the vector space (over \mathbb{Z}_3) $V = \mathbb{Z}_3^k$ for k such that $n = 3^k$. Our hash family will be the value-oblivious family defined as follows:

1. Choose a random non-zero vector $v \in V$.
2. Put each of the $\frac{n}{3}$ lines in V that is parallel to v in a set S independently with probability $\frac{n-1}{2m}$.
3. Choose a random hash function that causes collisions among all pairs of keys corresponding to points on the same line of some element of S , and has no other collisions.

In other words, we choose a hash function that causes collisions along many lines in a particular direction in our vector space. Note that the family of such hash functions is value-oblivious.

First, we would like to show that this hash family is 2-independent. Note that by Lemma (1) it is enough to show that for any two distinct keys p, q , the probability that they collide is $\frac{O(1)}{m}$. Notice that p and q can collide only if v is parallel to $p - q$. In other words, v must be $p - q$ or $2(p - q)$ (since we are working modulo 3, and since 1 and 2 are the only invertible elements in \mathbb{Z}_3). This happens with probability $\frac{2}{n-1}$. If v is one of these vectors, then p and q collide if and only if the line that they define is chosen to be in S . This happens with probability $\frac{n-1}{2m}$. Hence p and q collide with probability $\left(\frac{2}{n-1}\right) \left(\frac{n-1}{2m}\right) = \frac{1}{m}$. Hence this family is 2-independent.

Consider two hash functions, f and g , picked independently from this family. We will show that with the desired probability, there will exist some set of 9 elements that together f and g hash to only 8 distinct values, thus forcing the Cuckoo Hashing algorithm to rehash. We will show that this holds if we condition on any pair of v_f and v_g being the vectors chosen in step 1 of defining f and g , respectively.

Consider $\frac{n}{9}$ distinct planes in V that are parallel to both v_f and v_g . Suppose all three of the lines in any of these planes parallel to v_f are in S_f (the set from step 2 in defining f) and two of the three parallel to v_g are in S_g (the set from step 2 in defining g). Then f sends these 9 points to 3 distinct values and g sends them to at most 5, so together these are hashed to at most 8 distinct values.

This event happens with probability $3 \left(\frac{n-1}{2m}\right)^5 = O\left(\frac{n^5}{m^5}\right)$. Since this event happens independently for all $\frac{n}{9}$ planes, the probability that this event occurs for none of them is $\left(1 - O\left(\frac{n^5}{m^5}\right)\right)^{n/9}$. This proves our proposition. \square

A similar argument holds for 3-independence. In particular we will prove that

Proposition 4. *If n is a power of 2, there exists a 3-independent family of hash functions, so that if two functions are chosen independently from the family, Cuckoo Hashing with these hash functions avoids rehashing with probability $\left(1 - \Omega\left(\frac{n}{m}\right)^{10}\right)^{n/16}$. Therefore, the expected amortized time to hash n items with these families is at least $\exp\left(\Omega\left(\frac{n^{11}}{m^{10}}\right)\right)$.*

Proof. We associate our keys with elements of the vector space (over \mathbb{Z}_2) $V = \mathbb{Z}_2^k$ for k such that $n = 2^k$. Our hash family will be the value-oblivious family defined as follows:

1. Choose a random plane P in V .
2. Put each of the $\frac{n}{4}$ planes in V that is parallel to P in a set S independently with probability $O\left(\frac{n^2}{m^2}\right)$.
3. Choose a random hash function that causes collisions among all pairs of keys corresponding to points on the same plane of some element of S , and causes no other collisions.

In other words, we choose a hash function that causes collisions on many planes in a particular direction in our vector space. Note that the family of such hash functions is value-oblivious.

First, we would like to show that this hash family is 3-independent. Note that by Lemma (1) it is enough to show that for any two distinct keys p, q , the probability that they collide is $\frac{O(1)}{m}$, and for any three distinct keys, p, q, r , the

probability that they all collide is $\frac{O(1)}{m^2}$. Notice that p and q can collide only if P is parallel to $p - q$. This happens with probability $\frac{O(1)}{n}$. If P is parallel to $p - q$, then p and q collide if and only if the plane parallel to P through them is chosen to be in S . This happens with probability $O\left(\frac{n^2}{m^2}\right)$. Hence p and q collide with probability $O\left(\left(\frac{1}{n}\right)\left(\frac{n^2}{m^2}\right)\right) = \frac{O(n)}{m^2} = \frac{O(1)}{m}$. If p, q, r are distinct points, they all collide only if P is parallel to the plane defined by them (notice that such a plane is unique since they cannot be collinear due to the fact that we are working over \mathbb{Z}_2). This happens with probability $\frac{O(1)}{n^2}$. If P is parallel to this plane, then p, q, r collide if and only if the plane which passes through them is in S , which happens with probability $O\left(\frac{n^2}{m^2}\right)$. So the probability that p, q, r all collide is $\frac{O(1)}{m^2}$. Hence this family is 3-independent.

Consider two hash functions, f and g , picked independently from this family. We will show that with the desired probability, there will exist some set of 6 elements that together f and g hash to only 5 distinct values, thus forcing the Cuckoo Hashing algorithm to rehash. We will show that this holds if we condition on any pair of P_f and P_g being the planes chosen in step 1 of defining f and g , respectively.

Consider $\frac{n}{16}$ distinct 4-dimensional hyper-planes in V that are parallel to both P_f and P_g . Let H be one of these hyper-planes. Suppose that 3 of the planes in H parallel to P_f are in S_f , and two of the hyper-planes in H parallel to P_g are in S_g . Then each pair of these planes, one from S_f and the other from S_g , intersects at a point. This leads to a total of 6 points. On the other hand, these points are in the union of 3 planes in S_f and 2 in S_g , hence f and g hash them to only 5 distinct values.

This event happens with probability $O\left(\left(\frac{n^2}{m^2}\right)^5\right) = O\left(\frac{n^{10}}{m^{10}}\right)$. Since this event happens independently for all $\frac{n}{16}$ hyper-planes, the probability that this event occurs for none of them is $\left(1 - O\left(\frac{n^{10}}{m^{10}}\right)\right)^{n/16}$. This proves our proposition. □

4.4 Failure of 5-Independence

We prove the failure of Cuckoo Hashing for 5-independent hash families as follows: we describe a collision-space representation of a pair of 5-independent hash families and use a function from each family. We choose our collision-space representation to cause two connected cycles, which causes Cuckoo Hashing to rehash with high probability.

4.4.1 Overview of the Proof

We partition the keys in our universe into equally-sized sets, which we call *columns*. We then create a graph whose vertices are these columns and whose

edges form two connected cycles. Looking at each pair of adjacent columns, our hash functions will choose bijections between their elements to determine which elements collide. We will choose the bijections such that, if we compose them around each cycle, we will get the identity. We say that two keys a and b are in the same *level* if we can begin at a and apply some sequence of these bijections (or their inverses) to it going from column to adjacent column and eventually reach b . Note that being in the same level is an equivalence relation. Because composing bijections around a cycle gives the identity, each column has exactly one key on each level.

Our hash functions will cause collisions between an element x of a column c and the element of a column adjacent to c related to x by one of the bijections (or the inverse of one). For any pair of adjacent columns c and d , all collisions between their elements will be due to a particular one of our hash functions h_1 and h_2 . If any collisions between c and d are due to h_1 , none will be due to h_2 , and vice versa. This arrangement allows only for collisions along the same level. The graph of *potential collisions* (that is, a pair of keys in adjacent columns related by the corresponding bijection) on any level is the same as the graph on our columns.

We should clarify that we are considering two distinct graphs. One is the graph whose vertices are columns and whose edges define which columns are adjacent to one another. The other is the collision space graph whose vertices are the keys.

Each potential collision on any given level happens with probability $O(1)$ and they are independent of one another. We will have a fixed, finite number of columns (specifically, 32), so the probability that all potential collisions on a given level take place simultaneously is $O(1)$. Because we have $O(n)$ levels, with high probability all potential collisions on some level will occur, thus creating a pair of connected cycles and forcing Cuckoo Hashing to re-hash.

4.4.2 Description of the Bijections

We will associate with the keys in any column the members of a finite field F . The bijections between most pairs of columns will be a linear function on F , e.g. $X \rightarrow \alpha X + \beta$ where $\alpha, \beta \in F$ and $\alpha \neq 0$. Notice that the composition of two such functions is again such a function, and that the inverse of such a function is again such a function.

For two such bijections a, b we define their commutator $[a, b]$ to be $aba^{-1}b^{-1}$. The commutator of any two elements is a translation (i.e. it is of the form $X \rightarrow X + \beta$). Because any two translations commute, the commutator of two commutators is the identity. In particular, for bijections a, b, c, d of this form,

$$[[a, b], [c, d]] = aba^{-1}b^{-1}cdc^{-1}d^{-1}bab^{-1}a^{-1}dcd^{-1}c^{-1} = \text{Identity} \quad (3)$$

Our columns will be arranged into two cycles of length 16 with one edge e_0 between the cycles. The bijections corresponding to following edges around one

of the cycles beginning at an endpoint of e_0 will be, in order,

$$a_1, b_1, a_1^{-1}, b_1^{-1}, c_1, d_1, c_1^{-1}, d_1^{-1}, b_1, a_1, b_1^{-1}, a_1^{-1}, d_1, c_1, d_1^{-1}, c_1^{-1}$$

for randomly chosen bijections a_1, b_1, c_1, d_1 of the type described above. For the other cycle, we will use the same construction, but with four other random bijections a_2, b_2, c_2, d_2 . The bijection corresponding to e_0 will be a random bijection.

Notice that composing the bijections around a cycle gives the identity, due to equation (3). With bijections in the form of our bijections around the cycles, there is a unique bijection that sends x_0 to x_1 and y_0 to y_1 for any $x_0 \neq y_0, x_1 \neq y_1$.

4.4.3 Definition of the Hash Functions and Proof of their 5-Independence

We assume that $m = O(n)$ and n is 32 times the order of some finite field, F . We partition the keys into 32 columns as described above. For hash function h_1 we randomly pick bijections of the form described above a_1, a_2, c_1, c_2 . We pick those because we want each hash function to describe collisions corresponding to edges that are largely not incident to the same column because otherwise the analysis is more difficult (if it even works at all). Notice that the four bijections a_1, a_2, c_1, c_2 and their inverses only appear on consecutive edges of the cycles at the endpoints of e_0 (where we wrap around).

Note that because our bijections are between elements of F associated with keys, rather than with keys themselves, it is possible that two bijections will cause two keys from distinct columns corresponding to a value x from F to collide with two keys from other columns corresponding to ax . This could be too high a degree of correlation between the collisions (e.g. overly-dependent hash functions). To avoid this potential difficulty, we will restrict our hash functions so that this never occurs. The way we will do so is that for each $x \in F$ and each pair of adjacent columns whose associated bijection is a_1 or a_1^{-1} , we designate a random bijection as *owning* $\{x, a_1(x)\}$. Only that bijection will cause x to collide with $a_1(x)$ or $a_1(x)$ to collide with x . We will perform a similar operation with a_2, c_1 , and c_2 .

For each edge and each set that it owns, our hash function h_1 will cause a collision between the appropriate keys independently with probability $\Omega(1)$.

These will be all the collisions except in the cases where some element of a column collides with elements of two adjacent columns, in which case we collide those elements with each other, as well. For example, if x is at a column at an endpoint of e_0 and collides with $a_1(x)$ and $c_1(x)$, in the columns to its right and left, respectively, then in addition to those collisions, $a_1(x)$ and $c_1(x)$ collide, too. Notice that there cannot be any longer chains of collisions because nowhere is there a chain of more than two of a_1, a_2, c_1, c_2 on consecutive edges.

h_2 will be defined the same way as h_1 , except that h_2 will use the bijections b_1, b_2, d_1, d_2 and the random bijection corresponding to e_0 .

Lemma 5. *The hash families described by h_1 and h_2 are 5-independent.*

Proof. We begin by proving the statement for h_1 . The statement for h_2 is nearly analogous. Notice that because h_1 acts independently on both cycles of columns, it is enough to show 5-independence on one cycle of columns. We will make extensive use of lemma (1). Because no four elements can all collide, there are four requirements to prove: (1) given two distinct keys x and y , the probability that they collide is $\frac{O(1)}{m}$; (2) given three distinct keys x, y, z , the probability that they all collide is $\frac{O(1)}{m^2}$; (3) given two distinct pairs of keys (x_1, y_1) and (x_2, y_2) , the probability that each x_i collides with y_i is $\frac{O(1)}{m^2}$; and (4) given a pair (x_1, y_1) and a triplet (x_2, y_2, z_2) , the probability that the elements of the pair collide with one another and the elements of the triplet collide with one another is $\frac{O(1)}{m^3}$.

Case 1: pair colliding Call the pair (x, y) . If x and y are in adjacent columns in one cycle, they collide only if the bijection corresponding to the edge between the columns maps one to the other. This happens with probability $\frac{1}{|F|} = \frac{32}{n} = \frac{O(1)}{m}$. If they are not in adjacent columns in one cycle, the only way that they can collide is if they are both in columns adjacent to an endpoint of e_0 and the composition of the bijections mapping their columns to e_0 sends x to y . Because this composition is again a random bijection of the same form, this happens with probability $\frac{O(1)}{m}$.

Case 2: triplet colliding We can have a three-way collision only if the keys come from a column at the end of e_0 and both of its neighbors. Let x be the key at the column at the end of e_0 and let the others be y, z . Notice that a three-way collision only occurs if the bijections from x 's column to y and z 's columns map x to y and z , respectively. These each happen with probability $\frac{O(1)}{m}$ independently of each other, so the three-way collision occurs with probability $\frac{O(1)}{m^2}$.

Case 3: two pairs colliding Consider the bijections between the columns of x_1 and y_1 and between the columns of x_2 and y_2 . If these bijections are different and not inverses of one another, then we have these collisions if and only if one sends x_1 to y_1 and the other sends x_2 to y_2 . These events are independent of one another and each happens with probability $\frac{O(1)}{m}$, so both pairs collide with probability $\frac{O(1)}{m^2}$. If these bijections are the same (or inverses of one another) assume without loss of generality that the bijection from x_1 's column to y_1 's column is the same as the bijection from x_2 's column to y_2 's column. If x_1 and x_2 correspond to the same element of F or y_1 and y_2 correspond to the same element of F , then our discussion of owning pairs implies that at most one of these collisions will occur. Otherwise, these collisions occur only if the bijection between the associated columns sends x_i to y_i . This happens with probability $\frac{1}{|F|(|F|-1)} = \frac{O(1)}{m^2}$.

Case 4: pair and triplet colliding As in case 2, the triplet has to lie with one of its member's columns being an endpoint of e_0 and the other members being in columns adjacent to it. Note that the bijections between the columns of x_2 and y_2 , x_2 and z_2 , and y_2 and z_2 are, without loss of generality, $a_1, c_1,$

and $a_1^{-1}c_1$, respectively. We may also assume without loss of generality that the bijection between x_1 and y_1 is one of these three. We assume, without loss of generality, that this bijection is a_1 . Notice that if x_1 and x_2 correspond to the same element of F or y_1 and y_2 correspond to the same element of F then these collisions are impossible by the discussion of ownership. Otherwise, these collisions can occur only if a_1 maps x_1 and x_2 to y_1 and y_2 , respectively, and c_1 maps x_2 to z_2 . These events happen independently with probabilities $\frac{O(1)}{m^2}$ and $\frac{O(1)}{m}$, respectively. Hence these collisions happen with probability $\frac{O(1)}{m^3}$.

We find h_2 is 5-independent by an analogous argument, also noting that we need only show independence of h_2 separately on each of the cycles minus the endpoints of e_0 , and on the endpoints of e_0 . The arguments on the cycles are the same as the above, and the arguments on the endpoints of e_0 are analogous to subcases already discussed.

This completes our proof that the families corresponding to h_1 and h_2 are 5-independent. □

4.4.4 Probability of Rehashing

If we compose any number of the bijections (except all of them) going around any cycle, we get either a bijection with a random multiplicative component or a product of commutators which gives a random translation. Thus we get the identity as a product of some number of consecutive bijections (excluding the entire cycle) with probability $O(\frac{1}{n})$. If we compose all of the bijections, we get the identity. Because a non-identity bijection that is linear on F fixes at most one element, there are a finite number of elements that are fixed by these bijections. For all but finitely-many levels, all of the corresponding values in F to the keys on a given cycle of columns in the level are distinct. For such a level, each of the edges is owned by its column independently with probability $\Omega(1)$. Hence all of the edges between keys in such a level on adjacent columns collide with probability $\Omega(1)$.

Note that for any such level, there are only a fixed finite number of other levels that have edges that are competing with it for ownership of any pair of values in F . Therefore we can find $\Omega(n)$ levels that do not compete with one another for ownership. For each of these levels, all of the edges between keys in adjacent columns collide independently with probability $\Omega(1)$. Hence the probability that no level has all edges between keys in adjacent columns collide is $(1 - \Omega(1))^{\Omega(n)} = e^{-\Omega(n)}$, and the probability of not having to rehash is $O(\frac{1}{n} + e^{-\Omega(n)}) = O(\frac{1}{n})$.

Now all that we need to show is that when we rehash we do $\omega(1)$ work on average. Notice that we never rehash unless all of the elements of a given level have been hashed. We will look at elements of two adjacent columns and show that it takes $\omega(1)$ expected time before both elements on some level from those columns have been hashed. Consider the time when a elements from one column and b elements from the other have been hashed. Consider the expected number

of levels for which both elements in those columns have been hashed. Note that this is equal to the expected number of pairs of elements, one from the column containing a elements and one from the column containing b elements, that are on the same level. Note that for each of the ab pairs of elements, the probability that they are on the same level is $O(\frac{1}{n})$. Hence the expected number of levels containing one element from each set is $O(\frac{ab}{n})$. Hence if a and b are both $O(\sqrt{n})$ (with an appropriately small constant), then this expected value is less than $\frac{1}{2}$ and therefore the number of levels is zero with probability at least $\frac{1}{2}$. Therefore with probability at least $\frac{1}{2}$, we do not rehash until at least $\Omega(\sqrt{n})$ elements from at least one of the two columns have been hashed. Hence the expected time to rehash is $\Omega(\sqrt{n})$. Therefore the expected amount of time that Cuckoo Hashing takes to hash n elements is $\Omega(n^{\frac{3}{2}})$. This is because we must hash, on average, at least n times. This completes our proof that 5-independence is not sufficient for Cuckoo Hashing to work in expected amortized $O(1)$ time per operation.

4.5 A Matching Lower Bound for Joint-Independence

4.5.1 Overview of the Proof

Pagh and Rodler proved that Cuckoo Hashing works in expected $O(1)$ amortized time per operation when the hash functions are chosen to be $\Omega(\lg n)$ -independent [1]. Their proof does not use the full strength of independence, however, and in fact does not assume anything beyond $O(\lg n)$ -joint-independence. In the case of joint-independence, we will give a matching bound stating that anything below $\Omega(\lg n)$ -joint-independence can cause Cuckoo Hashing to fail to work in time, or even work at all.

We will show this fact by constructing a $\Theta(\lg n)$ -joint-independent joint-value-oblivious hash family whose collision-space representation for every pair of functions in the family has two large overlapping cycles which always cause Cuckoo Hashing to rehash when given the proper input.

4.5.2 Description of the Hash Functions

We choose hash families that are $c \lg n$ -joint-independent for some constant c such that $c \lg n$ is odd. c will be chosen later to be sufficiently small.

We begin by randomly selecting a set P of $2c \lg n + 1$ keys,

$$a_1, a_2, \dots, a_{c \lg n}, b_1, b_2, \dots, b_{c \lg n}, v.$$

These are the keys that will form our two cycles. v is the key at the intersection of the two cycles. We will choose the hash functions h_1 and h_2 so that h_1 collides v with a_1 and b_1 , and h_2 collides v with $a_{c \lg n}$ and $b_{c \lg n}$. h_1 collides a_1 with a_2 , h_2 collides a_2 with a_3 , and so on, and similarly with the b 's. There will be no other collisions between these keys and all other keys are assigned values randomly and independently.

There is a unique joint-value-oblivious hash family that has the above collision-space representation, and it will be the one chosen.

4.5.3 Proof that Cuckoo Hashing Fails

Note that h_1 maps the $2c \lg n + 1$ points to $c \lg n$ distinct values and h_2 maps them to the same number. Hence it is impossible to hash all of these to distinct values, and so cuckoo hashing is incapable of hashing all of these elements. Therefore the insertion algorithm can never terminate, and not only does Cuckoo Hashing fail to work in expected amortized $O(1)$ time per operation, it fails to work at all.

4.5.4 Proof of Joint-Independence

All that remains is to prove that the hash functions are $c \lg n$ -joint-independent. Let $C = c \lg n$. By lemma (2), it is sufficient to show that for any set T of C keys and any two equivalence relations, \sim_1 and \sim_2 , that split T into c_1 and c_2 equivalence classes, respectively, the probability that h_i collides any 2 elements in T that are related under \sim_i is $\frac{O(1)}{m^{2c-c_1-c_2}}$. Call the event just described E , for convenience.

We note that

$$\Pr(E) = \sum_{S \subset T} \Pr(E \cap (S = T \cap P)).$$

Consider the graph whose vertices are the elements of S and whose edges go between any pairs of vertices related by \sim_1 or \sim_2 . Suppose that this has g connected components. We want to look at all the ways that we can map S injectively into P such that the images of elements of S related under \sim_i collide in P under h_i . Note that this can only happen if for connected subsets of S , the number of connected components under \sim_1 plus the number of connected components under \sim_2 is the number of elements in the component plus 1. (This is true because the equivalent statement holds for connected subsets of P of size at most C .) Hence inscribing S into P is impossible unless the number of equivalence classes under \sim_1 plus the number of equivalence classes under \sim_2 is $|S| + g$.

Each connected component of S can be inscribed into P in $O(\lg n)$ different ways. (There are $O(\lg n)$ places to put a particular key and once one is fixed there are $O(1)$ ways to map the remainder of the component.) Hence the number of ways that we can map S into P is $O(\lg n)^g$. For each of these mappings of S into P , the probability that this is the actual correspondence of keys in T to keys in P is $\frac{O(1)}{n^{|S|}}$. Hence the probability that $S = T \cap P$ and that the collisions among elements of S are described by \sim_1 and \sim_2 is $\frac{O(\lg n)^g}{n^{|S|}}$.

Assuming that $S = T \cap P$ and the collisions among elements of S are described by \sim_1 and \sim_2 , we need to know the probability that the collisions in T are described by \sim_1 and \sim_2 . (We want to know this in order to compute the probability of E and $S = T \cap P$.) Consider the relations under \sim_1 . If we remove all but one of the vertices in each equivalence class of S , then the remaining vertices take values that are independent except that the values in S do not collide. Hence the probability of these collisions being described by \sim_1 is $O(1)$ divided by m to the power of the number of remaining vertices minus c_1 . We

have a similar relation for \sim_2 . Notice that the number of remaining vertices is equal to $C - |S|$ plus the number of equivalence classes of S under \sim_1 . Putting this all together, the probability that our collisions are described by \sim_1 and \sim_2 given that the collisions in S are described by \sim_1 and \sim_2 is $O(1)$ divided by m to the power of $2C - c_1 - c_2 - 2|S|$ plus the number of equivalence classes of S under \sim_1 plus the number of equivalence classes of S under \sim_2 . By the above, this last term is equal to $|S| + g$, hence this expression is

$$\frac{O(1)}{m^{2C - c_1 - c_2 - |S| + g}}.$$

Hence we have that

$$\begin{aligned} \Pr(E) &= \sum_{S \subset T} \Pr(E \cap (S = T \cap P)) \\ &\leq \sum_{S \subset T} \left(\frac{O(1)}{m^{2C - c_1 - c_2 - |S| + g}} \right) \left(\frac{(O(\lg n))^g}{n^{|S|}} \right) \\ &= \frac{O(1)}{m^{2C - c_1 - c_2}} + \sum_{S \subset T, S \neq \emptyset} \left(\frac{1}{m^{2C - c_1 - c_2}} \right) \left(\frac{O(\lg n)}{m} \right) \left(\frac{O(\lg n)}{m} \right)^{g-1} \left(\frac{m}{n} \right)^{|S|} \\ &\leq \frac{O(1)}{m^{2C - c_1 - c_2}} \left(1 + \left(\frac{O(\lg n)}{m} \right) \sum_{S \subset T, S \neq \emptyset} \left(\frac{m}{n} \right)^{|T|} \right) \\ &\leq \frac{O(1)}{m^{2C - c_1 - c_2}} \left(1 + \left(\frac{O(\lg n)}{m} \right) \left(\frac{2m}{n} \right)^C \right) \\ &= \frac{1}{m^{2C - c_1 - c_2}} \left(O(1) + \left(\frac{O(\lg n)}{m} \right) (O(1))^{c \lg n} \right) \\ &= \frac{O(1)}{m^{2C - c_1 - c_2}} \end{aligned}$$

The last line of the above holds if c is small enough. Therefore for sufficiently small c , our hash families are $c \lg n$ -joint-independent.

This completes our proof that $o(\lg n)$ -joint-independence is insufficient to guarantee that Cuckoo Hashing works.

5 Conclusion

We have shown that 5-independence of the hash families or less is insufficient to guarantee that Cuckoo Hashing will work in expected amortized $O(1)$ time per operation, and that if only joint-independence is guaranteed, the $O(\lg n)$ bound given by Pagh and Rodler [1] has a matching lower bound. Note that this matching bound means that any proof of an upper bound tighter than that given by Pagh and Rodler (if one exists) must use more than just the conditions of $o(\lg n)$ -joint-independence. Our lower bound of 5-independence

seems like it should be possible to improve, though none of our constructions actually improve upon it. We also showed one positive result, that only one of the hash functions need be $O(\lg n)$ independent; it is sufficient for the other to be 2-independent.

References

- [1] Rasmus Pagh and Flemming Friche Rodler, "Cuckoo Hashing", *Proceedings of the 9th European Symposium on Algorithms (ESA '01)* Springer-Verlag, 2001.