Bounds on the Independence Required for Cuckoo Hashing

Jeffrey Cohen and Daniel M. Kane

August 18, 2009

Abstract

Cuckoo Hashing is an efficient dynamic hashing technique. It uses two hash functions and hashes each key to the value indicated by either one of the hash functions. This is known to take expected $O(1)$ amortized time per operation as long as the hash functions are chosen independently and have at least $\Theta(\lg n)$-independence. It is not known whether this much independence is actually required, and this paper is an investigation of the degree of independence of the hash functions that is needed to achieve the same guarantee. In particular, we show that a pair of 5-independent hash functions will not necessarily suffice, among other results.

1 Introduction

Cuckoo Hashing is an efficient dynamic hashing technique [4]. It uses two hash functions and hashes each key to the value indicated by either one of the hash functions, giving two choices for the location of each key. This is known to take expected $O(1)$ amortized time per operation if the hash functions are sufficiently random. (Deletions and lookups are deterministically $O(1)$, but insertions are expected amortized $O(1)$.) Cuckoo Hashing's average performance is promising for practical applications, particularly after optimizations using a constant-sized stash [1] or de-amortization using a queue [2], [6].

The expected $O(1)$ amortized time bound per operation is known to hold as long as the hash functions are chosen independently and have at least $\Theta(\lg n)$-independence. The relevant asymptotics for purely random hash functions have been extensively analyzed by Kutzelnigg [5], but it remains unknown as to whether even $\Theta(\lg n)$ independence is actually required. Dietzfelbinger and Schellback have shown that Cuckoo Hashing fails to achieve its constant amortized time guarantee for some common 2-independent hash families, provided that the keys in the input are dense in the universe of possible keys [3]. This paper is an investigation of the degree of independence of the hash functions that will achieve the constant-time guarantee, and provides a stronger lower bound than previous results.

We will show that if the hash functions are both 5-independent, Cuckoo Hashing is not guaranteed to work in expected $O(1)$ amortized time per operation. We will show that if the hash functions are not chosen independently of one another but still satisfy a weaker condition that we call *joint-independence*, then both hash functions must exhibit $\Theta(\lg n)$-joint-independence or greater for Cuckoo Hashing to work within the same time bounds. Finally, we will demonstrate the power of choosing the hash functions independently of one another by proving that if one of the hash functions is $\Theta(\lg n)$-independent and the other is only 2-independent, Cuckoo Hashing still satisfies the same guarantee.

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 define *value-obliviousness,* a property of hash families that we will use in our proofs. In Section 4, we present an overview of Cuckoo Hashing. In Section 5 we show that 5-independent hash families are not necessarily sufficient for Cuckoo Hashing to work efficiently. In Section 6 we show that if the hash families are not chosen independently, but satisfy k-joint-independence, then $k \geq \Theta(\lg n)$ is both necessary and sufficient to guarantee that Cuckoo Hashing works efficiently. Finally, in Section 7, we show that if the hash functions are chosen independently, then only a single one of them needs to be $\Theta(\lg n)$-independent.

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 from a universe U to a set of values $\{1, 2, \ldots, m\}$ is a random way of choosing a hash function $f : U \to \{1, 2, \ldots, m\}$. The functions need not be chosen with equal probability.

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

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

This means that in a k-independent hash family, restricting the functions to a particular k keys, the function looks roughly like a random function on those keys.

Because this paper will consider *pairs* of hash functions, we also need a way to describe choices of random *pairs* of functions. Therefore we define a family of pairs of hash functions from a universe U to a set of values $\{1, 2, \ldots, m\}$ as a random way of choosing two hash functions $f, g : U \to \{1, 2, \ldots, m\}$.

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 sometimes refer to a family of pairs of hash functions as being k-independent for some k. By this we mean that it is a (k, k)-independent family. This idea is important because if the hash functions are correlated with

one another (for example, if they are always identical), then there is little hope of Cuckoo Hashing behaving efficiently.

We also define a weaker condition on families of pairs of hash functions that encapsulates the properties used in Pagh and Rodler's proof of the time bounds required for Cuckoo Hashing [4]. We say that a family of pairs of hash functions h_1, h_2 is *k-joint-independent* if for any k distinct keys $x_1, x_2, \ldots, x_k$ and $2k$ values $a_1, a_2, \ldots, a_k$ and $b_1, b_2, \ldots, b_k$, for a randomly chosen pair of hash functions,

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

In other words, a family of pairs of hash functions being k-joint-independent means that on any k keys the functions look roughly like a random pair of functions. Note that having (k, k)-independence is equivalent to having both k-joint-independence and independent selection of the hash functions.

3 Value-Obliviousness

We define a property of hash families called *value-obliviousness*. Every hash family we construct in this paper will be value-oblivious.

Definition. *A hash family is called* value-oblivious *if for any permutation π mapping the set of values $\{1, 2, \ldots, m\}$ to itself, the hash function h is chosen with the same probability as $\pi \circ h$.*

In value-oblivious hash families, there is a simple characterization of independence in terms of the collisions between keys.

Lemma 1. *A value-oblivious hash family is k-independent for $k = o(\sqrt{m})$ if and only if for any k distinct keys $x_1, \ldots, x_k$ and any equivalence relation, $\sim$, that splits $\{x_i\}$ into c equivalence classes, then for a randomly-chosen hash function h*

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

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, h, $h((a, b, c, \ldots)) = (h(a), h(b), h(c), \ldots)$. For any keys $x_1, \ldots, x_k$ and values $a_1, \ldots, a_k$, 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 $\sim$. The number of possible values of $\pi((a_1, a_2, \ldots, a_k))$ where π is some permutation of the values is $m(m-1)\ldots(m-c+1)$. Because the hash family is value-oblivious, rearranging the values does not alter the probability of their occurring. Thus,

$$\Pr(h((x_1, \ldots, x_k)) = (a_1, \ldots, a_k)) = \Pr(h((x_1, \ldots, x_k)) = \pi((a_1, \ldots, a_k))).$$

Since the left hand side of equation (1) is at least the sum of this probability over all possible values of $\pi((a_1, \ldots, a_k))$, we have that

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

By the definition of k-independence, this implies that the hash family is k-independent.

□

Definition. *A family of pairs of hash functions is called* joint-value-oblivious *if for any permutations π and ϕ mapping the set of values to itself, the hash functions (h_1, h_2) are chosen with the same probability as $(\pi \circ h_1, \phi \circ h_2)$.*

We have a similar characterization of joint-independence for joint-value-oblivious families in terms of their collisions:

Lemma 2. *A joint-value-oblivious family is k-joint-independent for $k = o(\sqrt{m})$ if and only if for any k distinct keys $x_1, \ldots, x_k$ and any equivalence relations, $\sim_l$, on the x_i that split them into c_l equivalence classes (for $l = 1, 2$), then for a randomly-chosen pair of hash functions (h_1, h_2)*

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

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

4 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 [4]. In Cuckoo Hashing, we choose hash functions h_1 and h_2 from two hash families and maintain two arrays. We guarantee that each key x in the input is hashed to either $h_1(x)$ in Array 1 or $h_2(x)$ in Array 2. If, while trying to insert element x, we find that element y_1 is in one of the spots where we might insert x, we put x in the other place where we might insert it. If we find that an element y_2 is in that place, we move one of the y_i to the other spot where we could place it. This may result in our needing a chain of replacements. If we get a chain that is too long or forms a loop (in which case we cannot eliminate all collisions), we pick new hash functions and try again. We call this *rehashing*.

Pagh and Rodler showed that Cuckoo Hashing performs insertions in expected $O(1)$ amortized time per insertion under the following conditions:

- The hash families are at least $\Theta(\lg n)$-independent

- Rehashing is performed after chains of $\Theta(\lg n)$ replacements
- The number of values, m, in each array is $\Theta(n)$ where n is the number of keys to be hashed
- The above bounds are chosen with appropriate constants

The times required for deletions and lookups are deterministically $O(1)$.

Note that the same proof works if, instead of requiring the families to be $\Theta(\lg n)$-independent, we only require them to be $\Theta(\lg n)$-join-independent.

5 Failure of 5-Independence

We prove the failure of Cuckoo Hashing for 5-independent hash families as follows: we describe a pair of 5-independent hash families and use a function from each family. We choose our families to cause loops of collisons which cause Cuckoo Hashing to rehash with high probability.

5.1 Overview of the Proof

We will cause Cuckoo Hashing to rehash with high probability by causing particular patterns of collisions which we call *cycles.*

Definition. *Suppose h_1 and h_2 are hash functions. A cycle is an ordered list of keys $(k_1, k_2, \ldots, k_{2j})$ for some positive integer $j \geq 1$ such that*

- $h_1(k_1) = h_1(k_2),\ h_1(k_3) = h_1(k_4),\ h_1(k_5) = h_1(k_6), \ldots$
- $h_2(k_2) = h_2(k_3),\ h_2(k_4) = h_2(k_5),\ h_2(k_6) = h_2(k_7), \ldots$
- $h_2(k_{2j}) = h_2(k_1)$.

In other words, each key in a cycle collides with the next key under alternating hash functions.

Lemma 3. *If any two cycles c_1 and c_2 contain disjoint sets of keys and there is a collision between any pair of keys, one from each cycle, then it is impossible to hash all of the keys in both of those cycles. Cuckoo Hashing will rehash if such an input is given.*

Proof. These two cycles have more keys than possible values that they could be mapped to. Therefore, by the pigeonhole principle, any assignment of keys to values will cause a collision. Therefore Cuckoo Hashing will rehash on any such input. □

Our goal is to choose a pair of hash families that generate overlapping cycles with high probability. We do this by dividing the keys into two sets, choosing hash functions that create a large number of cycles within each set, and then introducing collisions between the sets to make these cycles overlap.

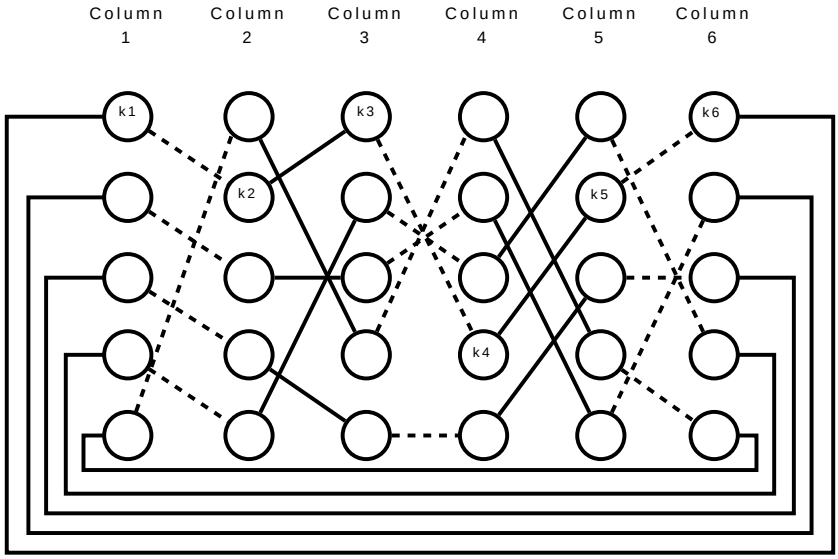

Figure 1: The keys in the figure are divided into six columns. (The actual hash functions will use twelve columns.) Dotted lines connect pairs of keys collided by one hash function, and solid lines connect pairs of keys collided by the other hash function. Each key in each column collides with exactly one key in the next column. Starting with any key k_1 in the leftmost column, let k_2 be the key in the next column it collides with, then k_3, and so on. The hash functions are chosen so that k_6 will collide with k_1 again, producing a cycle.

We will describe the way the hash families operate on one of the sets to create a large number of cycles. We group the keys in this set into twelve *columns*, $c_1, c_2, \ldots, c_{12}$. Hash functions in each family will cause collisions between keys in alternating adjacent pairs of columns. For example, hash functions in H_1 will cause collisions between keys in c_1 and keys in c_2. Hash functions in H_2 will cause collisions between keys in c_2 and keys in c_3. Hash functions in H_1 will cause collision between keys in c_3 and keys in c_4, and so on. Furthermore, each key in c_i will collide with a distinct key in c_{i+1} under the appropriate hash function, with keys in c_{12} colliding with keys in c_1 under H_2.

We put constraints on the collisions to ensure that cycles happen. Let k_1 be an arbitrary key in c_1 and let k_i be the key in c_i such that k_i collides with k_{i-1} under the appropriate hash function. Our hash families will be picked so that k_{12} collides with k_1. This creates a cycle. An illustration of such an arrangement of keys and collisions is shown in Figure 1.

Unfortunately, if we actually chose hash families that rigidly adhered to the above description, they would be too highly correlated, not 5-independent. Therefore our actual hash families will only produce a subset of these collisions. We include each of these collisions with some probability p. We will show that this construction results in a large number of cycles among the keys in the c_i

columns.

We will prove the following proposition:

Proposition 4. *For integers n, m, where $n \leq m$ and $p \in [0, 1]$ there exists a pair of hash families H_1 and H_2 mapping a set of keys of size n into an array of size m such that:*

- *If h_1 and h_2 are picked from H_1 and H_2, they lead to a number of cycles of size 12 given by the Bernoulli distribution with parameters q and p^{12} where q is at least $\frac{n}{24}$.*
- *H_1 and H_2 are 5-independent if $p = O\left(\frac{n}{m}\right)$*

5.2 Proof of Proposition 4

By Bertrand's Postulate, we can choose q to be a prime between $\frac{n}{24}$ and $\frac{n}{12}$. We proceed to throw away all but $12q$ of the keys. We split these keys into twelve columns of size q and associate the keys in each column with the elements of $\mathbb{Z}_q$ (the integers modulo q). For the rest of this proof, the term a^{-1} will refer to the multiplicative inverse of a in $\mathbb{Z}_q^*$. We will describe hash functions in the two families referred to in the proposition and show that they are 5-independent and have the required distribution of numbers of cycles.

H_1 is the hash family such that a member h_1 of it may be produced with the following procedure. Choose a random a in Z_q^* and a random $\alpha_0, \alpha_1, \alpha_2, \gamma_0, \gamma_1, \gamma_2$ from $\mathbb{Z}_q$ such that $\alpha_0 + \alpha_1 + \alpha_2 = \gamma_0 + \gamma_1 + \gamma_2 = 0$. For $i = 0, 1, 2$ and each key in c_{4i+1} corresponding to the element x in $\mathbb{Z}_q$, h_1 will collide this key with the key corresponding to $ax + \alpha_i$ in column c_{4i+2} with probability p. For each key in c_{4i+3} corresponding to the element y in $\mathbb{Z}_q$, h_1 will collide this key with the key corresponding to $a^{-1}y + \gamma_i$ in column c_{4i+4} with probability p. All of these probability p decisions are made independently.

H_2 is a hash family such that a member of it may be generated similarly, as follows.

Choose a random b in Z_q^* and a random $\beta_0, \beta_1, \beta_2, \delta_0, \delta_1, \delta_2$ from $\mathbb{Z}_q$ such that $\beta_0 + \beta_1 + \beta_2 = \delta_0 + \delta_1 + \delta_2 = 0$. For $i = 0, 1, 2$ and each key in c_{4i+2} corresponding to the element x in $\mathbb{Z}_q$, h_2 will collide this key with the key corresponding to $bx + \beta_i$ in column c_{4i+3} with probability p. For each key in c_{4i+4} corresponding to the element y in $\mathbb{Z}_q$, h_2 will collide this key with the key corresponding to $b^{-1}y + \delta_i$ in column c_{4i+5} (or c_1 if i is 2) with probability p. All of these probability p decisions are made independently.

Consider, for the moment, the case where $p = 1$ (that is, where all collisions that could happen given choices of a, b, α_i, β_i, etc do happen). Examine a key k_{4i+1} corresponding to the element $x_{4i+1} \in \mathbb{Z}_q$ in column c_{4i+1}. Under h_1, k_{4i+1} collides with k_{4i+2} in column c_{4i+2}. Similarly, k_{4i+2} collides with k_{4i+3} under h_2, which collides with k_{4i+4} under h_1, which collides with k_{4i+5} under h_2. Let k_j correspond to the element x_j in $\mathbb{Z}_q$. The value of x_{4i+5} is given by:

$$\begin{aligned}
x_{4i+5} &= b^{-1}x_{4i+4} + \delta_i \\
&= b^{-1}(a^{-1}x_{4i+3} + \gamma_i) + \delta_i \\
&= b^{-1}(a^{-1}(bx_{4i+2} + \beta_i) + \gamma_i) + \delta_i \\
&= b^{-1}(a^{-1}(b(ax_{4i+1} + \alpha_i) + \beta_i) + \gamma_i) + \delta_i \\
&= x_{4i+1} + a^{-1}\alpha_i + a^{-1}b^{-1}\beta_i + b^{-1}\gamma_i + \delta_i.
\end{aligned}$$

Pick k_1 in column c_1 corresponding to value $x_1 \in \mathbb{Z}_q$. Define k_{i+1} to be the column that k_i collides with in column c_{i+1} corresponding to $x_{i+1} \in \mathbb{Z}_q$. We call the key in c_1 that k_{12} collides with under h_2 by the name k_{13} and let it correspond to x_{13}. We have from the above that:

$$\begin{aligned}
x_{13} &= x_9 + a^{-1}\alpha_2 + a^{-1}b^{-1}\beta_2 + b^{-1}\gamma_2 + \delta_2 \\
&= x_5 + a^{-1}(\alpha_1 + \alpha_2) + a^{-1}b^{-1}(\beta_1 + \beta_2) + b^{-1}(\gamma_1 + \gamma_2) + (\delta_1 + \delta_2) \\
&= x_1 + a^{-1}(\alpha_0 + \alpha_1 + \alpha_2) + a^{-1}b^{-1}(\beta_0 + \beta_1 + \beta_2) \\
&\qquad + b^{-1}(\gamma_0 + \gamma_1 + \gamma_2) + (\delta_0 + \delta_1 + \delta_2) \\
&= x_1.
\end{aligned}$$

Therefore $k_{13} = k_1$. If $p = 1$, we have exactly q cycles (one for each key in column c_1).

Now, suppose p is not necessarily 1. Then not all of these q cycles will necessarily occur. We chose which collisions would happen by including each with probability p. Each of these possible cycles occurs if and only if each of the twelve collisions that comprise it was chosen to occur. This happens independently for each cycle with probability p^{12}, so the number of resulting cycles is a Bernoulli distribution with parameters q and p^{12}.

This proves the first half of Proposition 4. It remains to prove the second half, namely that for suitable values of p, H_1 and H_2 are five-independent. Assume that $p = O\left(\frac{n}{m}\right)$. H_1 and H_2 are defined and used symmetrically, so it is sufficient to only consider H_1.

H_1 is value-oblivious. Therefore it suffices to check the condition in Lemma 1. Because H_1 never collides a triple of elements, we need only check the probability that it collides a pair of given keys and the probability that it collides two pairs of given keys.

Consider one pair of keys. They will only collide if they are in a pair of consecutive columns collided by H_1. Without loss of generality, suppose the keys are k_1 and k_2 in columns c_1 and c_2 corresponding to elements $x_1, x_2 \in \mathbb{Z}_q$. Note that k_1 and k_2 will only collide if $x_2 = ax_1 + \alpha_0$. This happens with probability $\frac{1}{q}$. Furthermore, even if the above holds, the collision will only take place with probability p. Hence the probability that k_1 collides with k_2 is $\frac{p}{q}$, which is $O\left(\frac{n/m}{n}\right) = O\left(\frac{1}{m}\right)$.

Now, consider two pairs of keys. The pairs again have to be in appropriate pairs of consecutive columns. There are two cases: one where the pairs are in the same columns, and one where they are in different columns.

Suppose they are in the same pair of columns. Without loss of generality, suppose the pairs of keys are the ones corresponding to x_1 and y_1 in column c_1 and those corresponding to x_2 and y_2 in column c_2, with x_1 and x_2 colliding and y_1 and y_2 colliding. In order for these collisions to take place, it must be the case that $x_2 = ax_1 + \alpha_0$ and $y_2 = ay_1 + \alpha_0$. This happens with probability $\frac{1}{q(q-1)}$. Hence the probability of both collisions occurring is $\frac{p^2}{q(q-1)} = O\left(\left(\frac{n}{m}\right)^2 \left(\frac{1}{n}\right)^2\right) = O\left(\frac{1}{m^2}\right)$.

Now suppose they are in different pairs of columns. First consider the case where the pairs of keys are the ones corresponding to x_1 and x_2 in columns c_1 and c_2 and y_5 and y_6 in c_5 and c_6, with x_1 and x_2 colliding and y_5 and y_6 colliding. In order for these collisions to take place, it must be the case that $x_2 = ax_1 + \alpha_0$ and $y_5 = ay_6 + \alpha_1$. Note that α_0 and α_1 are independent from both each other and a. Therefore after fixing a each equations holds independently with probability $\frac{1}{q}$. Both hold with probability $\frac{1}{q^2}$. The collisions both take place with probability $\frac{p^2}{q^2} = O\left(\left(\frac{n}{m}\right)^2 \left(\frac{1}{n}\right)^2\right) = O\left(\frac{1}{m^2}\right)$.

The other cases of pairs of columns work similarly. After picking a, two of the α_i, γ_i must have particular values for the collisions to take place. Any two of these values are independent and chosen uniformly from among q possibilities, so all other pairs of columns work the same as the above case.

This proves that H_1 and H_2 are 5-independent if $p = O(\frac{n}{m})$, thus completing our proof of Proposition 4.

□

5.3 Description of the hash families and proof of their 5-independence

We now describe a pair of five-independent hash families, which we will show cause Cuckoo Hashing to fail. We call them F_1 and F_2.

We divide the keys into two disjoint sets, K and K', of size differing by at most one element. Let p be a constant that is $\Theta\left(\frac{n}{m}\right)$ as required by Proposition 4. Using Proposition 4 and this parameter p, we construct hash families H_1 and H_2 on K and H_1' and H_2' on K'.

The family F_1 is defined as follows. Pick an h_1 from H_1 and h_1' from H_1' randomly and independently from one another. Each key k in K is mapped to $h_1(k)$ and each key k' in K' is mapped to $h_1'(k')$. We define the family F_2 analogously.

Because F_1 is chosen by independently choosing members of a 5-independent family on K and a 5-independent family on K', F_1 is clearly 5-independent. The same is true of F_2.

5.4 Failure of Cuckoo Hashing

We now prove that our hash families defined above will produce a pair of overlapping cycles with high probability. We do this by first using Proposition 4 to show that there are many cycles consisting only of elements of K and many cycles consisting only of elements of K'. If any element of K in such a cycle collides with any element of K' in such a cycle, there are overlapping cycles and Cuckoo Hashing fails. We will show that this occurs with high probability as long as the number of keys n is large enough by proving the following theorem.

Theorem 5. *For $\epsilon > 0$, $n > m^{25/26+\epsilon}$, and $m > n$, Cuckoo Hashing with F_1 and F_2 fails with high probability.*

Proof. Recall from Proposition 4 that the number of cycles of size 12 consisting of elements in K is given by the Bernoulli distribution with parameters q and p^{12} where q is at least $\frac{n}{48}$ (we have 48 in the denominator instead of 24 because K contains only half the keys). The expected number of such cycles is $\Theta\left(n\left(\frac{n}{m}\right)^{12}\right) > \Theta(m^{1/2+\epsilon})$. By the Chernoff Bound, we get at least half this number with high probability. The same is true for K'.

When we defined H_1, we described how to choose the collisions that it causes among keys in K. It sends these keys to random values subject to that choice of collisions. A similar process holds for H_1' on K'. This still allows for F_1, which is based on H_1 and H_1', to collide keys in K with keys in K', because there is no restriction preventing a key in each set from mapping to the same value. Notice that if F_1 or F_2 sends a key in K involved in a cycle of length 12 and a key in K' involved in a cycle of length 12 to the same value, then Cuckoo Hashing will necessarily fail by Lemma 3.

We need to compute the probability that F_1 collides a key in K involved in a cycle of length 12 with a key in K' involved in a cycle of length 12. Suppose there are at least $\Theta(m^{1/2+\epsilon})$ cycles on each set. F_1 sends the keys in cycles on K' to $\Theta(m^{1/2+\epsilon})$ distinct elements. It sends the keys from cycles on K to $\Theta(m^{1/2+\epsilon})$ distinct elements that are chosen randomly and independently of each other and the locations of elements from K'. Suppose we pick the values to which we map keys in K for a fixed mapping of keys in K'. We do this by choosing where to map elements of K sequentially. When we choose to map the first element of K, the probability that it does not collide with an element of K' in such a cycle is $1 - \frac{\Theta(m^{1/2+\epsilon})}{m} = 1 - \Theta(m^{-1/2+\epsilon})$. Assuming that such a collision did not take place, the probability that the second element is in such a collision is $1 - \frac{\Theta(m^{1/2+\epsilon})}{m-1} < 1 - \Theta(m^{-1/2+\epsilon})$. Repeating this for all $\Theta(m^{1/2+\epsilon})$ keys, we find the probability that no such collision occurs is at most $\left(1 - \Theta(m^{-1/2+\epsilon})\right)^{\Theta(m^{1/2+\epsilon})} \leq e^{-\Theta(m^{2\epsilon})}$. Therefore with probability at least $1 - e^{-\Theta(m^{2\epsilon})}$, a collision will occur and Cuckoo Hashing will have to rehash.

Therefore, for these n insertions, Cuckoo Hashing will rehash at least an expected $e^{\Theta(m^{2\epsilon})} > \Theta(n)$ times, which means that it will not work in expected amortized constant time per operation.

□

6 A Matching Lower Bound for Joint-Independence

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 [4]. Their proof does not use the full strength of independence, however, and in fact does not assume anything beyond $\Omega(\lg n)$-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 in which every pair of functions has two large overlapping cycles which always cause Cuckoo Hashing to rehash when given the proper input.

We construct a family of pairs (h_1, h_2) of hash functions as follows:

Let k be an odd number such that $k = \Theta(\lg(n))$. Let P be a randomly chosen set of $2k+1$ keys. Randomly assign them the names $a_1, a_2, \ldots, a_k, v, b_1, b_2, \ldots, b_k$. Let h_1 collide the keys a_1, v, b_1, the pairs $(a_2, a_3), (a_4, a_5), \ldots$ and the pairs $(b_2, b_3), (b_4, b_5), \ldots$ and no other pair of keys. Let h_2 collide the triple a_k, v, b_k, the pairs $(a_1, a_2), (a_3, a_4), \ldots$ and the pairs $(b_1, b_2), (b_3, b_4), \ldots$ and no other pair of keys. Let h_1 and h_2 assign the keys to random values given these constraints. Note that this clearly defines a joint-value-oblivious family.

Given any such choice of h_1 and h_2, the keys in P are sent to only k distinct values by each h_i and hence cannot all be hashed to distinct values. This prevents Cuckoo Hashing from ever terminating on these inputs. We have left to show that this family is k-joint-independent.

For a set S of keys and equivalence relations $\sim_i$ on S where $i = 1, 2$, we let $c_i(S)$ be the number of equivalence classes of $\sim_i$ and let $E(S)$ be the event that

$$h_i(x) = h_i(y) \text{ for all } i = 1 \text{ or } 2, \text{ and for all } x, y \in S \text{ with } x \sim_i y.$$

By Lemma 2 it suffices to show that for any set S of keys with $|S| \leq k$, and any equivalence relations $\sim_i$ on S

$$\Pr(E(S)) = \frac{O(1)}{m^{2|S|-c_1(S)-c_2(S)}}. \tag{3}$$

Notice that the event $E(S)$ is dependent only on which elements of S correspond to which elements of P. When we choose our hash functions, for each $s \in S$, we either assign an element of P to s or we make s not correspond to any element in P. (Recall that P is the set of keys on which our collisions occur.) We are interested in the probability that for every x, y in S with $x \sim_i y$ that x and y correspond to elements of P that are collided by h_i. Note that if neither $\sim_i$ relates any distinct elements of S that (3) holds because $E(S)$ happens with probability 1 and $c_1(S) = c_2(S) = |S|$. Otherwise there is some subset $S_0 = \{x, y\}$ of S so that $x \sim_i y$ for some i. We can ignore the case where x and y are related by both $\sim_i$ because in that case, $E(S)$ is impossible.

Notice that for S_0 we have that:

$$\Pr(E(S_0)) \cdot m^{2|S_0|-c_1(S_0)-c_2(S_0)} = \frac{O(k)}{n^2} \cdot m^{2*2-1-2} = \frac{O(k)}{n^2} \cdot m = \frac{O(k)}{n}.$$

We will show that starting with S_0, we can add elements of S one at a time, building a subset S' of S, so that at each step we increase

$$\Pr(E(S')) \cdot m^{2|S'|-c_1(S')-c_2(S')} \tag{4}$$

by a factor of $O(1)$. By applying this $|S| - 2$ times, we discover that

$$\Pr(E(S)) \cdot m^{2|S|-c_1(S)-c_2(S)} = \frac{O(k) \cdot (O(1))^{O(k)}}{n} = O(1)$$

since $k \ll \log(n)$.

We begin with the set S_0 and we add elements to it. When choosing which elements to add, we always add an element that is related by an existing element under at least one $\sim_i$ if any such element exists. Otherwise, we add an arbitrary element. Call the set we've already built S', the new element x, and S' with x added S'_x. There are three cases for how x relates to the elements already in S', and we will now show that in each case, the value of (4) increases by at most a factor of $O(1)$.

Case 0: x is not related to any element in S' by either $\sim_i$
Clearly, $E(S') = E(S'_x)$. Notice that $2|S'| - c_1(S') - c_2(S')$ is unchanged by the addition of x because $|S'|$ and both $c_i(S')$ increase by 1. The value of (4) is unchanged by the addition of x.

Case 1: x is related to some elements in S' by exactly one of the $\sim_i$
Suppose, without loss of generality, that x is related to $y \in S'$ by $\sim_1$ but not related to any element of S' by $\sim_2$. This means that $c_1(S') = c_1(S'_x)$, while $c_2(S') + 1 = c_2(S'_x)$. Hence $2|S'| - c_1(S') - c_2(S')$ is 1 more than $2|S'_x| - c_1(S'_x) - c_2(S'_x)$. We wish to bound $\Pr(E(S'_x)|E(S'))$. Suppose that we've already decided which elements of S' correspond to which elements of P. There are at most two choices of elements of P that could be assigned to x that would cause $h_1(x) = h_1(y)$. Hence $\Pr(E(S'_x)|E(S')) \leq \frac{2}{n-|S'_x|} = \frac{O(1)}{n}$. Hence replacing S' by S'_x increases the value of (4) by a factor of at most $m \cdot \frac{O(1)}{n} = O(1)$.

Case 2: x is related to some elements in S' by both of the $\sim_i$
We will show that if this case ever occurs, $P(E(S')) = 0$. Suppose that $x \sim_1 y$ and $x \sim_2 z$ with y and z already in S'. Since x was added after both y and z, and since it is related to both of them, all of the elements added to S' since the first of y or z must have been related to some previous element (because we always add an element related to an existing element is possible). Therefore there must be a chain of elements of S' with each link related by either $\sim_1$ or $\sim_2$ connecting y to z. Adding the links $z \sim_2 x \sim_1 y$ produces a chain of elements which we call $w_1, w_2, \ldots, w_j$ where $j \leq k$ such that $w_1 \sim_1 w_2 \sim_2 \ldots \sim_1 w_j \sim_2 w_1$. (If the $\sim_i$'s in the chain do not alternate, we can choose a different chain in which

they do alternate by removing some of the elements.) Because h_1, h_2 produce no cycles of length k or less, $E(S')$ is impossible in this case. Hence replacing S' by S'_x makes (4) equal to 0.

Therefore each new element we add increases (4) by at most $O(1)$. Therefore $Pr(E(S))m^{2|S|-c_1(S)-c_2(S)} = O(1)$, so $Pr(E(S)) = \frac{O(1)}{m^{2|S|-c_1(S)-c_2(S)}}$. Therefore by Lemma (2) our function is $\Theta(\lg n)$-independent. This completes the proof.

7 Sufficiency of $(\Theta(\lg n), 2)$-Independence

Cuckoo Hashing is known to perform insertions, deletions, and lookups in expected amortized $O(1)$ time per operation for any $(\Theta(\lg n), \Theta(\lg n))$-independent hash family [4]. We now show that it works in $O(n)$ space and $O(1)$ expected amortized time for any $(\Theta(\lg n), 2)$-independent hash family with slight modification to Pagh and Rodler's algorithm[4]. Pagh and Rodler rehash when a chain of replacements of length $\Theta(\lg n)$ occurs. We also rehash when we've spent too much total time inserting since the last rehash. We do this because some hash functions are poor choices not because of long chains, but because of a large number of medium-length chains.

Recall the definition of a cycle from Section 5.1. A cycle is a sequence of keys $k_1, k_2, \ldots, k_\ell, k_1$ where each key collides with the next under alternating hash functions. We define a *path* similarly, except that the first and last keys don't necessarily need to be the same.

To prove our result, we will consider the cases which are problematic for Cuckoo Hashing and show that these are unlikely. These cases are based on comparing the lengths of paths and/or cycles to the degree of independence. We choose a constant $I = \Theta(\lg n)$ that is less than our degree of independence. The problematic cases are:

1. Any cycle of length at most I exists. (Such a cycle can cause a single insertion to fail.)

2. There exists any path of length at least I. (Such a path can cause a single insertion to fail.)

3. The sum of the lengths of all paths of length at most I is $\Omega(n)$ (If the previous case does not occur, this sum is an upper bound on the total number of swaps that may need to be done while inserting. If it is large, we may violate our runtime bounds.)

If none of these cases occur, we will be able to do our insertions in $O(n)$ time. Cases 1 and 2 not happening implies there are no cycles to make rehashing impossible. Cases 2 and 3 not happening implies that the sum of the lengths of chains of replacements is $O(n)$.

We will show that, for $m \geq cn$, for an appropriate constant c, we avoid any of these cases with probability at least $\frac{1}{2}$. We will rehash any time we have

performed more than $c_t n$ operations for some constant c_t since the last rehash. We will also rehash any time Pagh and Rodler do.

We will think of choosing the 2-independent hash function (h_2) first, then choosing 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 keys under h_2 (that is, the sets of keys hashing to the same value) $L_1, L_2, \ldots, L_k$. Let $A_i = |L_i|$ for each i. 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 at most I): Let ℓ be an integer such at 2ℓ is at most I. We want to consider the expected number of cycles whose keys lie in equivalence classes in a pattern like

$$L_{i_1}, L_{i_1}, L_{i_2}, L_{i_2}, \ldots, L_{i_\ell}, L_{i_\ell}.$$

(We repeat each index because every other pair of adjacent keys should be in the same equivalence class because they need to be collided under h_2. The other collisions are caused by $h_{\lg n}$ and go between equivalence classes.) The number of ways to pick distinct keys in these equivalence classes is at most $\prod_{j=1}^{\ell} A_{i_j}^2$. This is computed by counting the number of ways to pick two keys from each of the appropriate equivalent classes. For alternating pairs of keys to be collided by $h_{\lg n}$, we need ℓ pairs of keys to collide. Because $h_{\lg n}$ is more than 2ℓ-independent, the probability that such a sequence of keys is a cycle is $O(\frac{1}{m^\ell})$. Thus the probability that any of these sequences forms a cycle is at most $O\left(\prod_{j=1}^{\ell} \frac{A_{i_j}^2}{m}\right)$. The probability that we have any cycle of length 2ℓ is at most

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

Summing over all ℓ less than $\frac{I}{2}$ if C is sufficiently large, the expected number of cycles of any length at most I is at most $\frac{1}{8}$. Hence, with probability $\frac{7}{8}$, we have no cycles of these lengths.

Case 2 (paths of length at least I): Let ℓ be an integer such at 2ℓ is I or $I + 1$. From our above analysis, we see that the expected number of paths with vertices in equivalence classes in the pattern $L_{i_1}, L_{i_1}, L_{i_2}, L_{i_2}, \ldots, L_{i_\ell}, L_{i_\ell}$ is $O\left(m \prod_{j=1}^{\ell} \frac{A_{i_j}^2}{m}\right)$ (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 (because we are considering paths, not cycles). By the analysis in case (1), the expected number of paths of length $2\ell - 1$ of this form is $O\left(\frac{m}{C^\ell}\right)$.

For a sufficiently large C and ℓ being $\Theta(\lg n)$, this is at most $\frac{1}{8}$. Hence with probability at least $\frac{7}{8}$ there are no paths of length at least $2\ell - 1 \leq I$.

Case 3 (path-lengths sum to $\Omega(n)$): From our analysis of case 2, the expected number of paths of length $2\ell - 1 \leq I$ is $O\left(\frac{m}{C^\ell}\right)$. Hence the expected sum of the lengths of these paths is at most $O\left(\sum_{\ell=1}^{\infty} \ell \frac{m}{C^\ell}\right) = O(n)$. The probability that this sum is larger than some sufficiently large constant times n is at most $\frac{1}{8}$. Therefore with probability at least $\frac{7}{8}$ this case will not occur.

Assuming that the sum of the squares of the A_i is at most $\frac{m}{C}$, an event that happens with probability $\frac{7}{8}$, each of the three problematic cases is avoided with probability at least $\frac{7}{8}$. If the sum of the squares of the A_i is at most $\frac{m}{C}$, then the sum of the probabilities of the cases we wish to avoid is at most $\frac{3}{8}$. Therefore unconditionally, we avoid the unwanted cases with probability at least $\frac{7}{8}\left(1 - \frac{3}{8}\right) > \frac{1}{2}$ by the Union Bound.

We have now shown that with probability at least $\frac{1}{2}$ none of the three problematic cases occur. 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 insertions is $O(n)$.

This proves that Cuckoo Hashing will work in expected $O(1)$ amortized time per *insertion* on n *insertions* with appropriate choice of constants. We need to prove that Cuckoo Hashing will work in expected $O(1)$ amortized time per *operation*, even if we require that the amount of space allocated for the array cannot ever be more than a constant multiple of the number of keys stored.

To accomplish this, we need to slightly change how we rehash and choose array sizes. We will resize the arrays periodically. Initially they will be large enough to handle c insertions for a constant c. After $c/2$ operations, we will resize the arrays and rehash. Upon resizing, if there are m keys in the data structure, we will make the arrays large enough to handle $3m/2$ insertions. Then we will resize again after another $m/2$ operations and repeat ad infinitum.

The number of operations performed between resizings is $m/2$, and the expected total work is $O(3m/2)$ for the m reinsertions and the $m/2$ other operations. During this time, the size of the array is a constant times m, and the number of elements in the array is between $m/2$ and $3m/2$. Therefore we satisfy our time and space requirements.

8 Conclusion

We have given a proof that Cuckoo Hashing is not guaranteed to work in expected amortized $O(1)$ time per operation with 5-independent hash families. It seems likely that our construction does not yield a problem in any practical setting, because the actual numbers of elements required for it to fail are rather large and we had to precisely construct it to fail. However, these results suggest the possibility that a more typical hash family could fail.

We introduced the concept of joint-independent families and proved that Cuckoo Hashing cannot be guaranteed to work on based $o(\lg n)$ joint-independence

alone. Therefore, to improve the bound shown by Pagh and Rodler [4] one will need to use the stronger conditions of full independence. We used these conditions in our proof that only one family in Cuckoo Hashing need be $\Omega(\lg n)$-independent.

9 Acknowledgements

We would like to thank Professors Erik Demaine and David Karger and Mister Mihai Patrascu for their advice regarding this paper and for teaching the classes in which part of this material appeared as a final project. We would also like to thank Professor Jonathan Kane and Mister Michael Lieberman for their input on the language and organization of the paper.

References

[1] A. Kirsch, M. Mitzenmacher, and U. Wieder, "More Robust Hashing: Cuckoo Hashing with a Stash", *Proceedings of the 16th European Symposium on Algorithms (ESA '08)*, Springer-Verlag, 2008.

[2] A. Kirsch and M. Mitzenmacher, "Using a Queue to De-amortize Cuckoo Hashing in Hardware", *Proceedings of the 45th Annual Allerton Conference on Communication, Control, and Computing*, SIAM, 2007.

[3] Martin Dietzfelbinger, Ulf Schellbach, "On risks of using cuckoo hashing with simple universal hash classes", *ACM-SIAM Symposium on Discrete Algorithms (SODA '09)*, SIAM, 2009.

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

[5] Reinhard Kutzelnigg, "Bipartite Random Graphs and Cuckoo Hashing", *Proceedings of the Fourth Colloquium on Mathematics and Computer Science*, 2006.

[6] Yuriy Arbitman, Moni Naor, Gil Segev, "De-amortized Cuckoo Hashing: Provable Worst-Case Performance and Experimental Results", *36th International Colloquium on Automata, Languages and Programming (ICALP '09)*, Springer-Verlag, 2009.