More Robust Hashing: Cuckoo Hashing with a Stash*

Adam Kirsch† Michael Mitzenmacher‡ Udi Wieder§

Abstract

Cuckoo hashing holds great potential as a high-performance hashing scheme for real applications. Up to this point, the greatest drawback of cuckoo hashing appears to be that there is a polynomially small but practically significant probability that a failure occurs during the insertion of an item, requiring an expensive rehashing of all items in the table. In this paper, we show that this failure probability can be dramatically reduced by the addition of a very small constant-sized *stash*. We demonstrate both analytically and through simulations that stashes of size equivalent to only three or four items yield tremendous improvements, enhancing cuckoo hashing's practical viability in both hardware and software. Our analysis naturally extends previous analyses of multiple cuckoo hashing variants, and the approach may prove useful in further related schemes.

1 Introduction

In a multiple choice hashing scheme, each item can reside in one of d possible locations in a hash table. Such schemes allow for simple $O(1)$ lookups, since there are only a small number of places where an item can be stored. *Cuckoo hashing* refers to a particular class of multiple choice hashing schemes, where one can resolve collisions among items in the hash table by moving items as needed, as long as each item resides in one of its corresponding locations. Collisions, however, remain the bane of cuckoo hashing schemes and multiple choice hashing schemes in general: there is always some chance that on the insertion of a new item, none of the d choices are or can easily be made empty to hold it, causing a failure. In the theory literature, the standard response to this difficulty is to perform a full rehash if this rare event occurs. Since a failure in such schemes generally occurs with low probability (e.g., $O(n^{-c})$ for some constant $c \geq 1$), these rehashings have very little impact on the average performance of the scheme, but they make for less than ideal probabilistic worst case guarantees. Moreover, for many schemes, the constant c in the $O(n^{-c})$ failure probability bound is smaller than one actually desires in practice; values of $c \leq 3$ arguably lead to failures at too high a rate for commercial applications (assuming that the hidden constants are not too small). In particular, in many applications, such as indexing, elements are inserted and deleted from the hash table over a long period of time, increasing the probability of failure at some point throughout the life of the table. Furthermore, if the hash table is required to be *history independent* then a failure may trigger a long series of rehashings. See [10] for details.

*A conference version of this work appears in [7].

†School of Engineering and Applied Sciences, Harvard University. Supported in part by NSF grant CNS-0721491 and a grant from Cisco Systems. Email: kirsch@eecs.harvard.edu

‡School of Engineering and Applied Sciences, Harvard University. Supported in part by NSF grant CNS-0721491 and a grant from Cisco Systems. Email: michaelm@eecs.harvard.edu

§Microsoft Research Silicon Valley. Email: uwieder@microsoft.com

In this paper, we demonstrate that with standard cuckoo hashing variants, one can construct much more robust hashing schemes by utilizing very small amounts of memory outside the main table. Specifically, by storing a *constant* number of items outside the table in an area we call the *stash*, we can dramatically reduce the frequency with which full rehashing operations are necessary. A constant-sized stash is quite natural in most application settings. In software, one could use one or more cache lines for quick access to a small amount of such data; in hardware, one could effectively use content-addressable memories (CAMs), which are too expensive to store large tables but are cost-effective at smaller sizes. The intuition behind our approach is quite natural. If the items cause failures essentially independently, we should expect the number of items S that cause errors to satisfy $\mathbf{Pr}(S \geq s) = O(n^{-cs})$ for some constant $c > 0$ and every constant $s \geq 1$. In this case, if we can identify problematic items during the insertion procedure and store them in the stash, then we can dramatically reduce the failure probability bound.

Of course, failures do not happen independently, and formalizing our results requires revisiting and modifying the various analyses for the different variants of cuckoo hashing. We summarize our general approach. For many hashing schemes, it is natural to think of the hash functions as encoding a sample of a random graph G from some distribution. One can often show that the insertion procedure is guaranteed to be successful as long as G satisfies certain structural properties (e.g., expansion properties). The failure probability of the hashing scheme is then bounded by the probability that G does not satisfy these requirements. In this context, allowing a stash of constant size lessens these requirements, often dramatically reducing the corresponding failure probability. For example, if the properties of interest are expansion properties, then a stash effectively exempts sets of constant size from the expansion requirements. When such sets are the bottleneck in determining the failure probability, the stash allows dramatic improvements. Our work demonstrates that the technique of utilizing only a constant-sized stash is applicable to a number of interesting hashing schemes, and that one can often determine whether the technique is applicable by a careful examination of the original analysis. Furthermore, when the technique is applicable, the original analysis can often be modified in a fairly straightforward way.

Specifically, we first consider a variant of the cuckoo hashing scheme introduced by Pagh and Rodler [11], which uses two choices. We then consider variations proposed by Fotakis et al. [4], which utilizes more than two choices, and by Dietzfelbinger and Weidling [2], which allows buckets to hold more than one item. We verify the potential for this approach in practice via some simple simulations that demonstrate the power of a small stash.

Before continuing, we note that the idea of using a small amount of additional memory to store items that cannot easily be accommodated in the main hash table is not new to this work. For instance, Kirsch and Mitzenmacher [5, 6] examine hash table constructions designed for high-performance routers where a small number of items can be efficiently stored in a CAM of modest size. (In particular, [6] specifically considers improving the performance of cuckoo hashing variants by reordering hash table operations.) However, the constructions in [5] technically require a linear amount of CAM storage (although the hidden constant is very small), and the schemes in [6] are not formally analyzed. Our new constructions are superior in that they only require a small constant amount of additional memory and have provably good performance.

2 Standard Cuckoo Hashing

We start by examining the standard cuckoo hashing scheme proposed by Pagh and Rodler in [11]. Here we attempt to insert n items into a data structure consisting of two tables, T_1 and T_2, each with $m = (1+\epsilon)n$ buckets and one hash function (h_1 for T_1 and h_2 for T_2), where $\epsilon > 0$ is some fixed constant. Each bucket can store at most one item. To insert an item x, we place it in $T_1[h_1(x)]$ if that bucket is empty. Otherwise, we *evict* the item y in $T_1[h_1(x)]$, replace it with x, and attempt to insert y into $T_2[h_2(y)]$. If that location is free, then we are done, and if not, we evict the item z in that location and attempt to insert it into $T_1[h_1(z)]$, etc. Of course, this is just one variant of the insertion procedure; we could, in principle, attempt to place x in either of $T_1[h_1(x)]$ or $T_2[h_2(x)]$ before performing an eviction, or place an upper bound on the number of evictions that the insertion procedure can tolerate without generating some sort of failure. We find this variant simplest to handle.

Pagh and Rodler [11] show that if the hash functions are chosen independently from an appropriate universal hash family, then with probability $1 - O(1/n)$, the insertion procedure successfully places all n items with at most $\alpha \log n$ evictions for the insertion of any particular item, for some sufficiently large constant α. Furthermore, they show that if the insertion procedure is modified so that, if inserting a particular item requires more than $\alpha \log n$ evictions, the hash functions are resampled and all items in the table are reinserted, then the expected time required to place all n items into the table is $O(n)$.

Devroye and Morin [1] show that the success of the cuckoo hashing insertion procedure can be interpreted in terms of a simple property of a random multi-graph that encodes the hash functions[1]. In particular, Kutzelnigg [8] uses this approach to show that, if the hash functions are (heuristically) assumed to be independent and fully random, then the probability that the hash functions admit *any* injective mapping of the items to the hash buckets such that every item x is either in $T_1[h_1(x)]$ or $T_2[h_2(x)]$ is $1 - \Theta(1/n)$. (In fact, [8] identifies the exact constant hidden in the Theta notation.)

In this section, we use the approach of Devroye and Morin [1] to show that if the hash functions are independent and fully random and items that are not successfully placed in $\alpha \log n$ evictions result in some (easily found) item being placed in the stash, then the size S of the stash after all items have been inserted satisfies $\mathbf{Pr}(S \geq s) = O(n^{-s})$ for every integer $s \geq 1$. Equivalently, the use of a stash of constant size allows us to drive down the failure probability of standard cuckoo hashing exponentially.

We now proceed with the technical details. We view the hash functions h_1 and h_2 as defining a bipartite multi-graph with m vertices on each side, with the left and right vertices corresponding to the buckets in T_1 and T_2, respectively. For each of n items x, the hash values $h_1(x)$ and $h_2(x)$ are encoded as an instance of the edge $(h_1(x), h_2(x))$. Following [1], we call this multi-graph the *cuckoo graph*.

The key observation in [1] is that the standard cuckoo hashing insertion procedure successfully places all n items if and only if no connected component in the cuckoo graph has more than one cycle. In this case, the number of evictions required to place any item can be essentially bounded by the size of the largest connected component, which can be bounded with high probability using standard techniques for analyzing random graphs.

We modify the insertion algorithm in the following way: whenever an insertion of element x

[1]Some of the details in the proofs in [1] are not accurate and are corrected in part in this paper, as well as by Kutzelnigg [8].

fails, so the component of the cuckoo graph with the edge $(h_1(x), h_2(x))$ has more than one cycle, we put an item in the stash whose corresponding edge belongs to a cycle, effectively removing at least one cycle from the component. There are various ways of implementing an insertion algorithm with this property. One way is to observe that in a successful insertion, at most one vertex of the cuckoo graph is visited more than once, and no vertex is visited more than twice. Thus, if during an insertion we keep track of which memory slots we have already evicted items from, we can identify the slot that was evicted twice and thus put in the stash an element whose corresponding edge belongs to a cycle. This cycle detection mechanism requires us to remember how many times each slot was evicted. In practice, it may be better to set a limit of $\alpha \log n$ on the number of possible evictions. If $\alpha \log n$ evictions do not suffice then we 'roll back' to the original configuration (which we can do by remembering the last item evicted) and try to insert the element a second time, this time with a 'cycle detection' mechanism.

Of course, the most natural insertion algorithm is to impose an a-priori bound of $\alpha \log n$ on the number of evictions, and if after $\alpha \log n$ evictions an empty slot had not been found, put the current element in the stash. Unfortunately, this insertion algorithm does not guarantee that the element put in the stash corresponds to a cycle edge, a property essential for the analysis. Nevertheless, simulations given in Section 5 suggest that the same qualitative results hold for both cases.

The following theorem is the main result of this section.

Theorem 2.1. *For every constant integer $s \geq 1$, for a sufficiently large constant α, the size S of the stash after all items have been inserted satisfies $\mathbf{Pr}(S \geq s) = O(n^{-s})$.*

The rest of this section is devoted to the proof of Theorem 2.1. We start with the following observation, which is almost the same as one in [1].

Lemma 2.1. *Consider a walk W in the cuckoo graph corresponding to an insertion, and suppose that this walk takes place in a connected component of size k. Then the number of vertices visited during the walk (with multiplicity) is at most $k + 1$.*

Proof. From the definition of our insertion algorithm, W either contains no repeated vertices, or exactly one repeated vertex that occurs exactly twice. Since there are only k vertices in the connected component containing W, it is not possible for W to visit more than $k + 1$ vertices. □

The following observation allows us to quantify the relationship between the items that we put in the stash and the connected components in the cuckoo graph with at least two cycles.

Lemma 2.2. *Let G be a connected multi-graph with v vertices and $v + k$ edges, for some $k \geq 0$. Suppose that we execute the following procedure to completion: while G contains at least two cycles, we delete some edge in some cycle in G. Then the number of edges that we delete from G is exactly k.*

Proof. We use induction on k. For the base case note that if $k = 0$ then the graph has v nodes and v edges and is connected and therefore contains exactly one cycle. If $k \geq 1$ then the graph contains at least two cycles. When a cycle edge is removed the graph remains connected and has $v + (k - 1)$ edges. The induction hypothesis implies that the total number of edges removed is $1 + (k - 1) = k$ and the lemma follows. □

We are now ready to delve into the main technical details of the proof of Theorem 2.1. For a distribution D, let $\mathcal{G}(m, m, D)$ denote the distribution over bipartite graphs with m nodes on each

side, obtained by sampling $\ell \sim D$ and throwing ℓ edges independently at random (that is, each edge is put in the graph by uniformly and independently sampling its left node and its right node). Note that the cuckoo graph has distribution $\mathcal{G}(m, m, D)$ when D is concentrated at n. Now we fix some arbitrary vertex v of the $2m$ vertices. For any bipartite multi-graph G with m vertices on each side, we let $C_v(G)$ denote the connected component containing v. We then order the edges of G in some arbitrary way, and imagine that they are inserted into an initially empty graph in that order. We say that an edge is *bad* if at the time that it is inserted it closes a cycle (possibly of length 2). Note that while the set of bad edges depends on the ordering of the edges, the number of bad edges in each connected component of G is the same for all orderings. Thus, we may define $B_v(G)$ to be the number of bad edges in $C_v(G)$, and $f(G)$ to be the total number of bad edges in G. We also let $T(G)$ denote the number of connected components in G with at least one cycle.

Lemma 2.2 now tells us that S has the same distribution as $f(\mathcal{G}(m, m, n)) - T(\mathcal{G}(m, m, n))$. Thus, we have reduced the problem of bounding the size of the stash to the problem of analyzing the bad edges in the cuckoo graph. To that end we use stochastic dominance techniques.

Definition 2.1. For two graphs G and G' with the same vertex set V, we say that $G \geq G'$ if the set of edges of G contains the set of edges of G'. Similarly, for two tuples of graphs $(G_1, \ldots, G_t)$ and $(G'_1, \ldots, G'_t)$ with vertex set V, we say that $(G_1, \ldots, G_t) \geq (G'_1, \ldots, G'_t)$ if $G_i \geq G'_i$ for $i = 1, \ldots, t$. Let g be a function from t-tuples of graphs on V to reals. We say g is *non-decreasing* if $g(x) \geq g(y)$ whenever $x \geq y$.

Definition 2.2. Let μ and ν be two probability measures over t-tuples graphs with some common vertex set V. We say that μ *stochastically dominates* ν, written $\mu \succeq \nu$, if for every non-decreasing function g, we have $\mathbf{E}_\mu[g(G)] \geq \mathbf{E}_\nu[g(G)]$.

Since S has the same distribution as $f(\mathcal{G}(m, m, n)) - T(\mathcal{G}(m, m, n))$, and the function $f(G) - T(G)$ is increasing, it suffices to consider some distribution over graphs that stochastically dominates $\mathcal{G}(m, m, n)$. To this end, we let $\mathrm{Po}(\lambda)$ denote the Poisson distribution with parameter λ, or, where the context is clear, we slightly abuse notation by letting $\mathrm{Po}(\lambda)$ represent a random variable with this distribution. We now give the following stochastic dominance result.

Lemma 2.3. *Fix any $\lambda > 0$. For any $G \sim \mathcal{G}(m, m, \mathrm{Po}(\lambda))$, the conditional distribution of G given that G has at least n edges stochastically dominates $\mathcal{G}(m, m, n)$.*

Proof. For a left vertex u and a right vertex v, let $X(u, v)$ denote the multiplicity of the edge (u, v) in $\mathcal{G}(m, m, \mathrm{Po}(\lambda))$. By a standard property of Poisson random variables, the $X(u, v)$'s are independent with common distribution $\mathrm{Po}(\lambda/m^2)$. Thus, for any $k \geq 0$, the conditional distribution of G given that G has exactly k edges is exactly the same as $\mathcal{G}(m, m, k)$ (see, e.g., [9, Theorem 5.6]). Since $\mathcal{G}(m, m, k_1) \succeq \mathcal{G}(m, m, k_2)$ for any $k_1 \geq k_2$, the result follows. □

The key advantage of introducing $\mathcal{G}(m, m, \mathrm{Po}(\lambda))$ is the "splitting" property of Poisson distributions used in the proof of Lemma 2.3: if $\mathrm{Po}(\lambda)$ balls are thrown randomly into k bins, the joint distribution of the number of balls in the bins is the same as k independent $\mathrm{Po}(\lambda/k)$ random variables. This property simplifies our analysis. First, however, we must show that we can choose λ so that $\mathcal{G}(m, m, \mathrm{Po}(\lambda))$ has at least n edges with overwhelming probability for an appropriate choice of λ. This follows easily from a standard tail bound on Poisson random variables. Indeed, setting $\lambda = (1 + \epsilon')n$ for any constant $\epsilon' > 0$ gives

$$\mathbf{Pr}(\mathrm{Po}(\lambda) < n) \leq e^{-\lambda} \left(\frac{e\lambda}{n}\right)^n = e^{-n[\epsilon' - \ln(1+\epsilon')]} = e^{-\Omega(n)},$$

where we have used [9, Theorem 5.4] and the fact that $\epsilon' > \ln(1+\epsilon')$, which follows from the standard inequality $1+\epsilon' < e^{\epsilon'}$ for $\epsilon' > 0$. Therefore, by Lemmas 2.1 and 2.3,

$$\begin{aligned}\mathbf{Pr}(S \geq s) \leq \mathbf{Pr}(\max_v |C_v(\mathcal{G}(m,m,\mathrm{Po}(\lambda)))| > \alpha \log n) \\ + \mathbf{Pr}(f(\mathcal{G}(m,m,\mathrm{Po}(\lambda))) - T(\mathcal{G}(m,m,\mathrm{Po}(\lambda))) \geq s) + e^{-\Omega(n)}\end{aligned}$$

and so it suffices to show that for a sufficiently large constant α,

$$\mathbf{Pr}(\max_v |C_v(\mathcal{G}(m,m,\mathrm{Po}(\lambda)))| > \alpha \log n) = O(n^{-s}) \quad \text{and} \tag{1}$$

$$\mathbf{Pr}(f(\mathcal{G}(m,m,\mathrm{Po}(\lambda))) - T(\mathcal{G}(m,m,\mathrm{Po}(\lambda)) \geq s)) = O(n^{-s}). \tag{2}$$

Since we work with the probability space $\mathcal{G}(m,m,\mathrm{Po}(\lambda))$ from this point on, we slightly abuse notation and, for all vertices v, let $C_v = C_v(\mathcal{G}(m,m,\mathrm{Po}(\lambda)))$ denote the connected component containing v in $\mathcal{G}(m,m,\mathrm{Po}(\lambda))$, and let $B_v = B_v(\mathcal{G}(m,m,\mathrm{Po}(\lambda)))$ denote the number of bad edges in C_v. To establish (1), we first introduce a bound on $|C_v|$.

Lemma 2.4. *There exists some constant $\beta \in (0,1)$ such that for any fixed vertex v and integer $k \geq 0$, we have $\mathbf{Pr}(|C_v| \geq k) \leq \beta^k$.*

Proof. Let $X_1, X_2, \ldots$ be independent $\mathrm{Bin}(n, 1/m)$ random variables. By the exact same argument as in the proof of [1, Lemma 1], we have that for any $k \geq 1$,

$$\mathbf{Pr}(|C_v| \geq k) \leq \mathbf{Pr}\left(\sum_{i=1}^{k} X_i \geq k\right) = \mathbf{Pr}(\mathrm{Bin}(nk, 1/m) \geq k).$$

For $\epsilon \leq 1$, writing $nk/m = k/(1+\epsilon)$ and applying a standard Chernoff bound gives

$$\mathbf{Pr}(\mathrm{Bin}(nk, 1/m) \geq k) \leq e^{-\epsilon^2 k/3(1+\epsilon)} = \beta^k$$

for $\beta = e^{-\epsilon/3(1+\epsilon)} < 1$. To extend the proof to all $\epsilon > 0$, we simply note that $\mathbf{Pr}(\mathrm{Bin}(nk, 1/m) \geq k)$ is decreasing in ϵ for any fixed k and n. □

Clearly, Lemma 2.4 establishes (1) for a sufficiently large constant α. Turning our attention to (2), we first bound the number of bad edges in a single connected component of $\mathcal{G}(m,m,\mathrm{Po}(\lambda))$, and then use a stochastic dominance argument to obtain a result that holds for all connected components. Then we have the following key technical lemma.

Lemma 2.5. *For every vertex v and $t, k, n \geq 1$,*

$$\mathbf{Pr}(B_v \geq t \mid |C_v| = k] \leq \left(\frac{3e^5 k^3}{m}\right)^t.$$

Proof. We reveal the edges in C_v following a breadth-first search starting at v. That is, we first reveal all of the edges adjacent to v, then we reveal all of the edges of the form (u,w) where u is a neighbor of v, and so on, until all of C_v is revealed. Suppose that during this process, we discover that some node u is at distance i from v. Define $B(u)$ to be the number of edges that connect u to nodes at distance $i-1$ from v. In other words, $B(u)$ is the number of edges that connect u to the

connected component containing v at the time that u is discovered by the breadth-first search. It is easy to see that $B_v = \sum_u \max\{0, B(u) - 1\}$. We bound B_v by bounding $B(u)$ for each u.

Fix some $i \geq 1$, condition on the history of the breadth-first search until the point where all nodes with distance at most $i - 1$ from v have been revealed, and suppose that the size of the connected component containing v at this point in time is at most k. Then any node u that is not currently in the connected component containing v could be reached from at most k distinct vertices in the component in the next step of the breadth-first search procedure. Thus $B(u)$ is stochastically dominated by the sum of k independent $\mathrm{Po}(\lambda/m^2)$ random variables, which has distribution $\mathrm{Po}(k\lambda/m^2)$. (Here we are using the property that throwing $\mathrm{Po}(\lambda)$ edges randomly into the bipartite graph is the same as setting the multiplicity of the edges to be independent $\mathrm{Po}(\lambda/m^2)$ random variables.) For any $\epsilon' \leq \epsilon$, we have $k\lambda/m^2 \leq k/m$, and so $\mathrm{Po}(k\lambda/m^2)$ is stochastically dominated by $\mathrm{Po}(k/m)$. We conclude that the number of bad edges incident on u that are revealed at this point in the breadth-first search is stochastically dominated by the distribution $L(k) = \max(0, \mathrm{Po}(k/m) - 1)$. Furthermore, since the number of occurrences of each edge are independent, the joint distribution of the number of bad edges introduced at this point in the breadth-first search is stochastically dominated by sum of m independent samples from $L(k)$. It follows that the distribution of B_v given $|C_v| = k$ is stochastically dominated by the sum of mk independent samples from $L(k)$. We derive a tail bound on this distribution in Lemma 2.6 below, which yields the desired result. □

Lemma 2.6. *Fix $k \leq 2m$, and let $X_1, \ldots, X_{mk}$ be independent random variables with common distribution $L(k)$, and let $X = \sum_{i=1}^{mk} X_i$. Then for every $t \geq 1$, we have $\mathbf{Pr}(X \geq t) \leq \left(\frac{3e^5k^3}{m}\right)^t$.*

Proof. First we bound the number of the X_i's that are greater than zero. For every i we have $\mathbf{Pr}(X_i > 0) \leq \mathbf{Pr}(\mathrm{Po}(k/m) \geq 2) \leq k^2/m^2$ (since for any $\mu > 0$, we have $\mathbf{Pr}(\mathrm{Po}(\mu) \leq 1) = e^{-\mu}(1 + \mu) \geq (1 - \mu)(1 + \mu) = 1 - \mu^2$). The number of positive X_i's is therefore stochastically dominated by the binomial distribution $\mathrm{Bin}(mk, k^2/m^2)$. Let $P = \{i : X_i > 0\}$ denote the set of positive X_i's. We have

$$\mathbf{Pr}(X \geq t) \leq \mathbf{Pr}[|P| > t] + \sum_{\ell=1}^{t} \mathbf{Pr}(|P| \geq \ell) \cdot \mathbf{Pr}\left(\sum_{i \in P} X_i \geq t \;\middle|\; |P| = \ell\right) \tag{3}$$

We bound the first term by

$$\mathbf{Pr}(|P| \geq \ell) \leq \binom{mk}{\ell} \left(\frac{k^2}{m^2}\right)^\ell \leq \left(\frac{mke}{\ell}\right)^\ell \left(\frac{k^2}{m^2}\right)^\ell = \left(\frac{ek^3}{m\ell}\right)^\ell \tag{4}$$

For the second term, let $Y \sim \mathrm{Po}(k/m)$, and note that for every $j \geq 0$,

$$\begin{aligned}
\mathbf{Pr}(X_i = j + 1 \mid i \in P) &= \mathbf{Pr}(Y = j + 2 \mid Y \geq 2) \\
&\leq \frac{\mathbf{Pr}(Y = j + 2)}{\mathbf{Pr}(Y = 2)} \\
&= \frac{2m^2 e^{k/m}}{k^2} \mathbf{Pr}(Y = j + 2) \\
&\leq \frac{2e^2 m^2}{k^2} \mathbf{Pr}(Y = j + 2).
\end{aligned}$$

Now let $Y_1 \dots Y_\ell$ be independent random variables with common distribution $\mathrm{Po}(k/m)$. Then

$$\begin{aligned}
\mathbf{Pr}\left(\sum_{i\in P} X_i \geq t \;\middle|\; |P| = \ell\right) &= \sum_{\substack{j_1,\dots,j_\ell \\ \sum_{i=1}^{\ell} j_i \geq t-\ell}} \prod_{i=1}^{\ell} \mathbf{Pr}(X_i = j_i + 1 \mid X_i > 0) \\
&\leq \left(\frac{2e^2m^2}{k^2}\right)^{\ell} \sum_{\substack{j_1,\dots,j_\ell \\ \sum_{i=1}^{\ell} j_i \geq t-\ell}} \prod_{i=1}^{\ell} \mathbf{Pr}(Y_i = j_i + 2) \\
&\leq \left(\frac{2e^2m^2}{k^2}\right)^{\ell} \mathbf{Pr}\left(\sum_{i=1}^{\ell} Y_i \geq t + \ell\right) \\
&= \left(\frac{2e^2m^2}{k^2}\right)^{\ell} \mathbf{Pr}(\mathrm{Po}(k\ell/m) \geq t + \ell) \\
&\leq \left(\frac{2e^2m^2}{k^2}\right)^{\ell} \left(\frac{ek\ell}{m(t+\ell)}\right)^{t+\ell} \\
&= \frac{1}{m^{t-\ell}} \left(\frac{2e^3\ell}{k(t+\ell)}\right)^{\ell} \left(\frac{ek\ell}{t+\ell}\right)^{t}, \qquad (5)
\end{aligned}$$

where we have used the tail bound [9, Theorem 5.4] in the fifth step.

Substituting (4) and (5) into (3) yields

$$\begin{aligned}
\mathbf{Pr}(X \geq t) &\leq \left(\frac{ek^3}{mt}\right)^{t} + \sum_{\ell=1}^{t} \left(\frac{ek^3}{m\ell}\right)^{\ell} \cdot \frac{1}{m^{t-\ell}} \left(\frac{2e^3\ell}{k(t+\ell)}\right)^{\ell} \left(\frac{ek\ell}{t+\ell}\right)^{t} \\
&= \left(\frac{ek^3}{mt}\right)^{t} + \left(\frac{ek}{m}\right)^{t} \sum_{\ell=1}^{t} \left(\frac{2e^4k^2}{t+\ell}\right)^{\ell} \left(\frac{\ell}{t+\ell}\right)^{t} \\
&\leq \left(\frac{ek^3}{mt}\right)^{t} + \left(\frac{ek}{m}\right)^{t} 2e^4k^2 \\
&\leq \left(\frac{ek^3}{m}\right)^{t} + \left(\frac{2e^5k^3}{m}\right)^{t} \\
&\leq \left(\frac{3e^5k^3}{m}\right)^{t},
\end{aligned}$$

completing the proof. □

Combining Lemmas 2.5 and 2.4 now tells us that for any vertex v and constant $t \geq 1$,

$$\begin{aligned}
\mathbf{Pr}(B_v \geq t) &\leq \sum_{k=1}^{\infty} \mathbf{Pr}(B_v \geq t \mid |C_v| = k) \cdot \mathbf{Pr}(|C_v| \geq k) \\
&\leq \sum_{k=1}^{\infty} \left(\frac{3e^5k^3}{m}\right)^{t} \cdot \beta^k \\
&= O(n^{-t}) \quad \text{as } n \to \infty. \qquad (6)
\end{aligned}$$

Equation (6) gives a bound for the number of bad edges in a single connected component of $G(m, m, \mathrm{Po}(\lambda))$. We now extend this result to all connected components in order to show (2), which will complete the proof. The key idea is the following stochastic dominance result.

Lemma 2.7. *Fix some ordering $v_1, \dots, v_{2m}$ of the vertices. For $i = 1, \dots, 2m$, let $C'_{v_i} = C_{v_i}$ if v_i is the first vertex in the ordering to appear in C_v, and let C'_{v_i} be the empty graph on the $2m$ vertices otherwise. Let $C''_{v_1}, \dots, C''_{v_m}$ be independent random variables such that each C''_{v_i} is distributed as C_{v_i}. Then $(C''_{v_1}, \dots, C''_{v_{2m}})$ stochastically dominates $(C'_{v_i}, \dots, C'_{2m})$.*

Proof. We prove the result by showing that there exists a coupling where the C''_{v_i}'s have the appropriate joint distribution and C'_{v_i} is a subgraph of C''_{v_i} for $i = 1, \dots, 2m$. We do this by showing how to sample the relevant random variables so that all of the required properties are satisfied. First, we simply sample C_{v_1} and set $C''_{v_1} = C'_{v_1} = C_{v_1}$. Then, for the smallest i such that $v_i \notin C_{v_1}$, we set $C'_{v_2}, \dots, C'_{v_{i-1}}$ to be the empty graph and sample $C'_{v_i} = C_{v_i}$ according to the appropriate conditional distribution. Note that this conditional distribution can be represented as the connected component containing v_i in a sample from a distribution over bipartite graphs with vertex set $\{v_1, \dots, v_{2m}\} - C_{v_1}$, where for each left vertex u and right vertex v, the multiplicity of the edge (u, v) is a $\mathrm{Po}(\lambda/m^2)$ random variable, and these multiplicities are independent. This conditional distribution is clearly stochastically dominated by C_{v_i}, and therefore we can sample C_{v_i} according to the correct conditional distribution and simultaneously ensure that C_{v_i} is a subgraph of C''_{v_i} and that the distribution of C''_{v_i} is not affected by conditioning on C''_{v_1}. We continue in this way, next sampling $C'_{v_j} = C_{v_j}$ and C''_{v_j} for the smallest $j \notin C_{v_1} \cup C_{v_i}$ according to the appropriate conditional distribution and setting $C'_{v_{i+1}}, \dots, C'_{v_{j-1}}$ to be the empty graph, etc., until we have sampled all of the C'_{v_i}'s. At this point, we have that C'_{v_i} is a subgraph of C''_{v_i} for every i such that C'_{v_i} is not the empty graph, and we have not yet sampled the C''_{v_i}'s for which C'_{v_i} is the empty graph. To complete the construction, we simply sample these remaining C''_{v_i}'s independently from the appropriate distributions. □

Now let B denote the common distribution of the B_v's, and let $B'_1, \dots, B'_{2m}$ be independent samples from B. By Lemma 2.7, we have that $f(\mathcal{G}(m, m, \mathrm{Po}(\lambda))) - T(\mathcal{G}(m, m, \mathrm{Po}(\lambda)))$ is stochastically dominated by $\sum_{i=1}^{2m} B'_i - |\{i : B'_i \geq 1\}|$. Applying (6) now implies that there exists a constant

$c \geq 1$ such that for sufficiently large n,

$$
\begin{aligned}
&\mathbf{Pr}(f(\mathcal{G}(m,m,\mathrm{Po}(\lambda))) - T(\mathcal{G}(m,m,\mathrm{Po}(\lambda))) \geq s) \\
&\leq \mathbf{Pr}\left(\sum_{i=1}^{2m} B'_i \geq s + |\left\{i \,:\, B'_i \geq 1\right\}|\right) \\
&\leq \sum_{\substack{j_1,\ldots,j_{2m} \\ \sum_{i=1}^{2m} j_i = s}} \prod_{\substack{i=1,\ldots,2m \\ j_i \geq 1}} \mathbf{Pr}(B \geq j_i + 1) \\
&= \sum_{\substack{j_1,\ldots,j_{2m} \\ \sum_{i=1}^{2m} j_i = s}} \prod_{\substack{i=1,\ldots,2m \\ j_i \geq 1}} cn^{-j_i-1} \\
&\leq \sum_{\substack{j_1,\ldots,j_{2m} \\ \sum_{i=1}^{2m} j_i = s}} c^s n^{-s-|\{i\,:\,j_i \geq 1\}|} \\
&= \sum_{k=1}^{2m} \sum_{\substack{j_1,\ldots,j_{2m} \\ \sum_{i=1}^{2m} j_i = s \\ |\{i\,:\,j_i \geq 1\}| = k}} c^s n^{-s-k} \\
&\leq \sum_{k=1}^{2m} \binom{2m}{k} k^s c^s n^{-s-k} \\
&\leq \sum_{k=1}^{2m} \left(\frac{2(1+\epsilon)ne}{k}\right)^k k^s c^s n^{-s-k} \\
&= n^{-s} c^s \sum_{k=1}^{2m} \left(\frac{2e(1+\epsilon)}{k}\right)^k k^s \\
&= O(n^{-s}),
\end{aligned}
$$

which establishes (2), completing the proof of Theorem 2.1.

3 Generalized Cuckoo Hashing

We now turn our attention to the generalized cuckoo hashing scheme proposed by Fotakis et al. [4]. Here, we attempt to insert n items into a table with $(1+\epsilon)n$ buckets and d hash functions (assumed to be independent and fully random), for some constant $\epsilon > 0$. We think of the hash functions as defining a bipartite random multi-graph model $\mathcal{G}(n, \epsilon, d)$, which is sampled by creating n left vertices, representing items, each with d incident edges, and $(1+\epsilon)n$ right vertices, representing hash locations. The right endpoints of the edges are chosen independently and uniformly at random from the vertices on the right. We think of a partial placement of the items into the hash locations as a matching in $\mathcal{G}(n, \epsilon, d)$. For a graph G in the support of $\mathcal{G}(n, \epsilon, d)$ and a matching M in G, we let G_M denote the directed version of G where an edge e is oriented from right to left if $e \in M$, and e is oriented from left to right if $e \notin M$.

To perform an insertion of one of the n items, we think of the current placement of items into hash locations as defining a matching M on a sample G from $\mathcal{G}(n, \epsilon, d)$, and then we simulate a

breadth-first search of depth at most $2t + 1$ on G_M starting from the left vertex u corresponding to the new item, for some $t \geq 0$ to be specified later. If we encounter an unmatched right vertex v during this process, then we move the items in the hash table accordingly to simulate augmenting M using the discovered path from u to v. If not, then we declare the insertion procedure to be a failure.

Fotakis et al. [4] show the following three results, which we extend to a variant of the insertion procedure that uses a stash.

Proposition 3.1. *For any constant $\epsilon \in (0, 1)$ and $d \geq 2(1 + \epsilon)\ln(e/\epsilon)$, a sample G from $\mathcal{G}(n, \epsilon, d)$ contains a left-perfect matching with probability $1 - O(n^{4-2d})$.*

Proposition 3.2. *For any $d < (1 + \epsilon)\ln(1/\epsilon)$, the probability that a sample G from $\mathcal{G}(n, \epsilon, d)$ contains a left-perfect matching is $2^{-\Omega(n)}$.*

Theorem 3.1. *It is possible to choose $t = O(\ln(1/\epsilon))$ such that for any constants $\epsilon \in (0, 0.2)$ and $d \geq 5 + 3\ln(1/\epsilon)$, the probability that the insertion of the n items completes without generating a failure is $O(n^{4-d})$ as $n \to \infty$.*

Proposition 3.1 is essentially a feasibility result, in that it tells us that it is highly likely that the hash functions admit a valid placing of the items into the table, for an appropriate choice of $d = \Omega(\ln(1/\epsilon))$. Proposition 3.2 tells us that this lower bound on d is asymptotically tight. Theorem 3.1 then tells us that for appropriate ϵ and d, not only do the hash functions admit a valid placing of the items into the table with high probability, but the insertion algorithm successfully finds such a placement by using a breadth-first search of depth $O(\ln(1/\epsilon))$.

Finally, we note that the emphasis of [4] is slightly different from ours. That work also shows that, with high probability, no insertion operation requires the examination of more than $o(n)$ right vertices with high probability. It also shows that, if, whenever a failure occurs, the hash functions are resampled and all items in the table are reinserted, then the expected time to insert a single item is $O(\ln(1/\epsilon))$. While these are significant results, they follow fairly easily from the analysis used to prove Theorem 3.1, and the exact same arguments apply to the counterpart to Theorem 3.1 that we prove later in this section, which considers a variation of the insertion procedure that allows for items to be placed in a stash. Thus, for our purposes, Theorem 3.1 is the most significant result in [4], and so we use it as our benchmark for comparison.

It is important to recall that, in practice, one would not expect to use a breadth first search for placement, but instead use a random walk approach, replacing a random one of the choices for the item to be placed at each step [4]. Analyzing this scheme (even without a stash) remains an important open problem. Of course, in our experiments in Section 5, we consider this more practical variant with the stash.

Having reviewed the results of [4], we are now ready to describe a way to use a stash in the insertion procedure. The modification is very simple: whenever an insertion operation for an item x would generate a failure during the original procedure, we attempt to reinsert every item currently in the stash into the table, and then we add x into the stash. Alternatively, if there is some maximum size s of the stash, then if inserting an item x into the table using the original procedure would result in a failure, we simply place x in the stash if the stash has fewer than s items, and otherwise we attempt to reinsert every item in the stash into the table, until (hopefully) one of those insertions succeeds. In that case, we can place x in the stash, and otherwise we declare a failure. This variant is probably better suited to practice, since it only requires us to attempt to

reinsert all items in the stash when the stash is full. However, the first method is easier to work with (since it never generates a failure), so we use it in the following discussion, although our results can be applied to the second method as well.

Let S denote the maximum size of the stash as the n items are inserted. We show the following three results, which should be viewed as counterparts to Proposition 3.1, Proposition 3.2, and Theorem 3.1, respectively.

Proposition 3.3. *For any constants $c, \epsilon > 0$, for sufficiently large constant d, for every integer constant $s \geq 0$, the probability that a sample G from $\mathcal{G}(n, \epsilon, d)$ does not have a matching of size at least $n - s$ is $O(n^{1-c(s+1)})$ as $n \to \infty$. Furthermore, the minimum value of d necessary for this result to hold is at most $d = (2 + o(1)) \ln(1/\epsilon)$, where here the asymptotics are taken as $\epsilon \to 0$ with c held constant.*

Proposition 3.4. *For every constant $\epsilon > 0$, $s \geq 0$, and $d \leq (1 + \epsilon) \ln\left(\frac{1+\epsilon}{2(\epsilon+s/n)}\right)$, the probability that a sample G from $\mathcal{G}(n, \epsilon, d)$ contains a matching of size $n - s$ is $2^{-\Omega(n)}$.*

Theorem 3.2. *For every constants $c > 0$ and $\epsilon \in (0, 0.2)$, for sufficiently large constant d, for every integer constant $s \geq 1$, we have $\mathbf{Pr}(S \geq s) = O(n^{1-cs})$ as $n \to \infty$. Furthermore, the minimum value of d necessary for this result to hold is at most $3 \ln(1/\epsilon) + O(1)$, where here the asymptotics are taken as $\epsilon \to 0$ with c held constant.*

Like Proposition 3.1, Proposition 3.3 tells us that for an appropriate choice of $d = \Omega(\ln(1/\epsilon))$, it is likely that the hash functions admit a placing of at least $n - s$ items into the table and at most s items into the stash. Proposition 3.4 then tells us that this lower bound on d is asymptotically tight. Finally, Theorem 3.2 tells us that with a stash of bounded, constant size, our modified insertion algorithm gives a dramatically improved upper bound on the failure probability for inserting the items when compared to Theorem 3.1 for the original insertion algorithm, for the same number of hash functions.

The remainder of this section is devoted to the proofs of Proposition 3.3, Proposition 3.4, and Theorem 3.2. Since the proof of Proposition 3.4 is very easy, we prove it first, using essentially the same technique as Fotakis et al. [4] use to prove Proposition 3.2.

Proof of Proposition 3.4. We bound the probability p that G has at most $\epsilon n + s$ isolated right vertices, since otherwise a maximal matching in G has size less than $n - s$. This probability p is the same as the probability that throwing nd balls randomly into $(1 + \epsilon)n$ bins yields at most $\epsilon n + s$ empty bins. By a standard Poisson approximation lemma (e.g. [9, Theorem 5.10]), we have

$$\begin{aligned} p &\leq 2\mathbf{Pr}\left(\mathrm{Bin}\left((1+\epsilon)n, \mathbf{Pr}\left(\mathrm{Po}\left(\frac{nd}{(1+\epsilon)n}\right) = 0\right)\right) \leq \epsilon n + s\right) \\ &= 2\mathbf{Pr}\left(\mathrm{Bin}\left((1+\epsilon)n, e^{-d/(1+\epsilon)}\right) \leq \epsilon n + s\right). \end{aligned}$$

Now,

$$\mathbf{E}\left[\mathrm{Bin}\left((1+\epsilon)n, e^{-d/(1+\epsilon)}\right)\right] = (1+\epsilon)ne^{-d/(1+\epsilon)} \geq 2(\epsilon n + s),$$

and so Hoeffding's inequality gives

$$p \leq 2\exp\left[-\frac{(\epsilon + s/n)^2 n}{2(1+\epsilon)}\right] = 2^{-\Omega(n)},$$

as required. □

It remains to show Proposition 3.3 and Theorem 3.2. The proof technique here is essentially the same as in [4]. As in that work, the key idea is that a sample G from $\mathcal{G}(n, \epsilon, d)$ satisfies certain left-expansion properties with high probability. By Hall's theorem, these properties are sufficient to guarantee the existence of a matching of the appropriate size, yielding Proposition 3.3. For Theorem 3.2, we show that if G satisfies these expansion properties, then the insertion procedure is guaranteed to keep the size of the stash bounded by a fixed constant s. Thus, for both results, the error probability arises entirely from the possibility that G may not satisfy all of the desired left-expansion requirements.

Unfortunately, it appears that to prove our results in this way, we must repeat a large portion of the analysis in [4]. Thus, while there may appear to be a great deal of work necessary to prove our stash results, we are really just performing a few clever modifications to the proofs in [4]. We emphasize that we consider this property to be a strength of this work, as it bolsters our argument that our stash techniques can be easily incorporated into many hashing schemes and subsequently analyzed without a great deal of additional insight.

Returning to the proofs of Proposition 3.3 and Theorem 3.2, we begin by giving a number of technical lemmas concerning the expansion properties of $\mathcal{G}(n, \epsilon, d)$.

Lemma 3.1 ([4, Proposition 1])**.** *For integers $1 \le k \le n$,*

$$\binom{n}{k} \le \left(\frac{n}{n-k}\right)^{n-k} \left(\frac{n}{k}\right)^{k}.$$

Lemma 3.2. *Fix some $k \in \{1, \dots, n\}$ and $\epsilon \in (0, 1]$, $\gamma \ge 0$, and $c > 0$. Let $\mu = k/n$, and choose d so that*

$$d \ge f(\mu, \epsilon, \gamma) \triangleq 1 + \gamma + c + \frac{\mu \ln \frac{1}{\mu} + (1-\mu) \ln \frac{1}{1-\mu} + (1 + \epsilon - (1+\gamma)\mu) \ln \frac{1+\epsilon}{1+\epsilon-(1+\gamma)\mu}}{\mu \ln \frac{1+\epsilon}{(1+\gamma)\mu}}.$$

Then the probability that there exists a set of k left vertices in a sample G from $\mathcal{G}(n, \epsilon, d)$ with at most $(1+\gamma)k$ neighbors is at most

$$\left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{ck}.$$

Furthermore, if we set $\gamma = \delta\epsilon$ for some constant $\delta \in [0, 1)$, then as $\epsilon \to 0$, we have

$$\sup_{\mu \in (0,1]} f(\mu, \epsilon, \gamma) \le (2 + o(1)) \ln \frac{1}{(1-\delta)\epsilon}.$$

Similarly, as $\epsilon \to 0$ with γ and $\delta \in (0, 1)$ held constant, $\sup_{\mu \in (0, \delta/(1+\gamma)]} f(\mu, \epsilon, \gamma) = O(1)$.

Proof. We bound the probability $p(k)$ that there is a set of k left vertices with at most $(1+\gamma)k$ neighbors using a union bound over all sets X of k left vertices and all sets Y of $(1+\gamma)k$ right vertices of the event that all neighbors of X are in Y. Thus, we have

$$\begin{aligned}
p(k) &\le \binom{n}{k}\binom{(1+\epsilon)n}{(1+\gamma)k}\left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{dk} \\
&\le \left(\frac{n}{n-k}\right)^{n-k}\left(\frac{n}{k}\right)^{k}\left(\frac{(1+\epsilon)n}{(1+\epsilon)n-(1+\gamma)k}\right)^{(1+\epsilon)n-(1+\gamma)k}\left(\frac{(1+\epsilon)n}{(1+\gamma)k}\right)^{(1+\gamma)k}\left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{dk} \\
&= \left[\left(\frac{1}{1-\mu}\right)^{1-\mu}\left(\frac{1}{\mu}\right)^{\mu}\left(\frac{(1+\epsilon)}{(1+\epsilon)-(1+\gamma)\mu}\right)^{1+\epsilon-(1+\gamma)\mu}\left(\frac{1+\epsilon}{(1+\gamma)\mu}\right)^{(1+\gamma)\mu}\left(\frac{(1+\gamma)\mu}{1+\epsilon}\right)^{\mu d}\right]^{n},
\end{aligned}$$

and so $p(k) \leq \left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{ck}$ as long as $d \geq f(\mu, \epsilon, \gamma)$.

Now we need two facts from [4].

Lemma 3.3 ([4, Proposition 2]). *For any $\alpha \in (0,1)$ and $\beta \in (0,1]$,*

$$\frac{(1-\beta)\ln\frac{1}{1-\beta}}{\beta\ln\frac{1+\alpha}{\beta}} \leq \frac{\alpha\ln\frac{1+\alpha}{\alpha}}{\ln(1+\alpha)}.$$

The following lemma is taken from [4].

Lemma 3.4. *For any fixed $\alpha \in (0,1)$,*

$$\frac{(1+\alpha-x)\ln\frac{1+\alpha}{1+\alpha-x}}{x\ln\frac{1+\alpha}{x}}$$

is a non-decreasing function of x in the interval $(0,1]$.

We are now ready to bound $f(\mu, \epsilon, \gamma)$ under the assumption that $0 < \gamma < \epsilon \leq 1$. First, we note that $\mu\ln(1/\mu) < \mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}$. Applying Lemma 3.3 with $\alpha = \frac{\epsilon-\gamma}{1+\gamma} \in (0,1)$ and $\beta = \mu$ gives

$$\frac{(1-\mu)\ln\frac{1}{1-\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} \leq \frac{\frac{\epsilon-\gamma}{1+\gamma}\ln\left(\frac{1+\epsilon}{\epsilon-\gamma}\right)}{\ln\frac{1+\epsilon}{1+\gamma}}.$$

Also, by Lemma 3.4 with $\alpha = \frac{\epsilon-\gamma}{1+\gamma} \in (0,1)$ (and $1+\alpha = \frac{1+\epsilon}{1+\gamma}$),

$$\begin{aligned}
\frac{(1+\epsilon-(1+\gamma)\mu)\ln\frac{1+\epsilon}{1+\epsilon-(1+\gamma)\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} &= (1+\gamma)\frac{\left(\frac{1+\epsilon}{1+\gamma}-\mu\right)\ln\frac{\frac{1+\epsilon}{1+\gamma}}{\frac{1+\epsilon}{1+\gamma}-\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} \\
&\leq (1+\gamma)\frac{\left(\frac{1+\epsilon}{1+\gamma}-1\right)\ln\frac{\frac{1+\epsilon}{1+\gamma}}{\frac{1+\epsilon}{1+\gamma}-1}}{\ln\frac{1+\epsilon}{1+\gamma}} \\
&= \frac{(\epsilon-\gamma)\ln\frac{1+\epsilon}{\epsilon-\gamma}}{\ln\frac{1+\epsilon}{1+\gamma}}.
\end{aligned}$$

Therefore,

$$f(\mu,\epsilon,\gamma) \leq 2+\gamma+c+\frac{\frac{\epsilon-\gamma}{1+\gamma}\ln\frac{1+\epsilon}{\epsilon-\gamma}+(\epsilon-\gamma)\ln\frac{1+\epsilon}{\epsilon-\gamma}}{\ln\frac{1+\epsilon}{1+\gamma}} = 2+\gamma+c+\frac{(\epsilon-\gamma)(2+\gamma)\ln\frac{1+\epsilon}{\epsilon-\gamma}}{(1+\gamma)\ln\frac{1+\epsilon}{1+\gamma}},$$

which does not depend on μ. Next, we examine this upper bound on $f(\mu, \epsilon, \gamma)$ as $\epsilon \to 0$, for $\gamma = \delta\epsilon$,

where $\delta \in [0,1)$ is a fixed constant. Indeed, using Taylor series, we have that as $\epsilon \to 0$,

$$\begin{aligned}
\frac{(\epsilon-\gamma)(2+\gamma)\ln\frac{1+\epsilon}{\epsilon-\gamma}}{(1+\gamma)\ln\frac{1+\epsilon}{1+\gamma}} &= \frac{\epsilon(1-\delta)(2+\delta\epsilon)\ln\frac{1+\epsilon}{(1-\delta)\epsilon}}{(1+\delta\epsilon)\ln\frac{1+\epsilon}{1+\delta\epsilon}} \\
&= \frac{\epsilon(1-\delta)(2+\delta\epsilon)\left(\ln(1+\epsilon)+\ln\frac{1}{(1-\delta)\epsilon}\right)}{(1+\delta\epsilon)\left(\ln(1+\epsilon)-\ln(1+\delta\epsilon)\right)} \\
&= \frac{\epsilon(1-\delta)(2+\delta\epsilon)\left(O(\epsilon)+\ln\frac{1}{(1-\delta)\epsilon}\right)}{(1+\delta\epsilon)\left((\epsilon+O(\epsilon^2))-(\delta\epsilon+O(\epsilon^2))\right)} \\
&= \frac{(1-\delta)(2+\delta\epsilon)\left(O(\epsilon)+\ln\frac{1}{(1-\delta)\epsilon}\right)}{(1+\delta\epsilon)\left(1-\delta+O(\epsilon)\right)} \\
&= (2+o(1))\ln\frac{1}{(1-\delta)\epsilon}.
\end{aligned}$$

Next, we bound $f(\mu,\delta,\epsilon)$ under the assumption that $0<\mu\le\delta/(1+\gamma)$ for some $\delta\in(0,1)$ and $\epsilon<\gamma$. First, we note that

$$\frac{\mu\ln\frac{1}{\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} = \frac{1}{1-\frac{\ln\frac{1+\gamma}{1+\epsilon}}{\ln\frac{1}{\mu}}}$$

is an increasing function of μ, and therefore

$$\frac{\mu\ln\frac{1}{\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} \le \frac{\ln((1+\gamma)/\delta)}{\ln((1+\epsilon)/\delta)} = O(1) \quad \text{as } \epsilon\to 0.$$

Second, we note that

$$\frac{(1-\mu)\ln\frac{1}{1-\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} = \frac{\frac{1}{\mu}\ln\frac{1}{1-\mu}}{\frac{1}{1-\mu}\ln\frac{1+\epsilon}{(1+\gamma)\mu}},$$

and that the numerator of the latter expression is increasing in μ, while the denominator is decreasing. Thus,

$$\frac{(1-\mu)\ln\frac{1}{1-\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} \le \frac{(1+\gamma)\left(1-\frac{\delta}{1+\gamma}\right)\ln\frac{1}{1-\frac{\delta}{1+\gamma}}}{\delta\ln((1+\epsilon)/\delta)} = O(1) \quad \text{as } \epsilon\to 0.$$

Third, we apply Lemma 3.4 with $\alpha=\epsilon$ and $x=(1+\gamma)\mu\in(0,\delta)$ to obtain

$$\begin{aligned}
\frac{(1+\epsilon-(1+\gamma)\mu)\ln\frac{1+\epsilon}{1+\epsilon-(1+\gamma)\mu}}{\mu\ln\frac{1+\epsilon}{(1+\gamma)\mu}} &= (1+\gamma)\frac{(1+\epsilon-x)\ln\frac{1+\epsilon}{1+\epsilon-x}}{x\ln\frac{1+\epsilon}{x}} \\
&\le (1+\gamma)\frac{(1+\epsilon-\delta)\ln\frac{1+\epsilon}{1+\epsilon-\delta}}{\delta\ln\frac{1+\epsilon}{\delta}} = O(1) \quad \text{as } \epsilon\to 0.
\end{aligned}$$

Combining the last three bounds gives $\sup_{\mu\in(0,\delta/(1+\gamma)]} f(\mu,\epsilon,\gamma) = O(1)$ as $\epsilon\to 0$, completing the proof. □

Lemma 3.5. *For any $\epsilon, \gamma, c > 0$ and $k_1 \leq k_2 \in \{1, \ldots, n\}$,*

$$\sum_{k=k_1}^{k_2} \left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{ck} \leq (k_2 - k_1 + 1) \max_{k \in \{k_1, k_2\}} \left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{ck}.$$

Proof. We examine the function $g : [k_1, k_2] \to \mathbb{R}$ given by

$$g(x) = \left(\frac{(1+\gamma)x}{(1+\epsilon)n}\right)^{cx}.$$

It is easy to see that $g(x)$ is convex, we are raising a linear function to the cx power. It follows that $g(x)$ is at maximized at either $x = k_1$ or $x = k_2$. The result follows. □

Lemma 3.6. *Let $1 \geq \epsilon > \gamma \geq 0$, $c > 0$, and $d \geq \sup_{\mu \in (0,1]} f(n, \epsilon, \gamma)$ as in Lemma 3.2. Fix any $k^* \geq 1$. As $n \to \infty$, the probability that a sample G from $\mathcal{G}(n, \epsilon, d)$ contains a set S of at least k^* left vertices with fewer than $(1+\gamma)|S|$ neighbors is $O\left(n^{1-ck^*}\right)$.*

Proof. By Lemma 3.2 and a union bound, the probability of interest is at most

$$\sum_{k=k^*}^{n} \left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{ck}.$$

The result now follows directly from Lemma 3.5. □

Lemma 3.7. *Let $\epsilon \in (0, 1]$, $\gamma \geq 0$, $c > 0$, $\delta \in (0, 1)$, and $d \geq \sup_{\mu \in (0, \delta/(1+\gamma)]} f(\mu, \epsilon, \gamma)$ as in Lemma 3.2. Fix some constant $k^* \geq 1$. The probability that a sample G from $\mathcal{G}(n, \epsilon, d)$ contains a set S of left vertices with $k^* \leq |S| \leq \delta n/(1+\gamma)$ and fewer than $(1+\gamma)|S|$ neighbors is $O\left(n^{1-ck^*}\right)$.*

Proof. By Lemma 3.2 and a union bound, the probability of interest is at most

$$\sum_{k=k^*}^{\delta n/(1+\gamma)} \left(\frac{(1+\gamma)k}{(1+\epsilon)n}\right)^{ck}.$$

The result now follows directly from Lemma 3.5. □

Lemma 3.8 ([4, Lemma 3])**.** *For constant $\epsilon \in (0, 0.2)$ and $d \geq 5 + 3\ln(1/\epsilon)$, the probability that every set of right vertices Y in $\mathcal{G}(n, \epsilon, d)$ with $\epsilon n \leq |Y| \leq 3n/8$ has at least $4|Y|/3$ neighbors is at least $1 - 2^{\Omega(n)}$ as $n \to \infty$.*

We remark that a version of Lemma 3.8 could probably be proven directly from Lemma 3.2 without too much trouble, but since we are in a position to quote it directly from [4], there is no point in reproving it.

We now return to the proofs of Proposition 3.3 and Theorem 3.2. Indeed, we now have more than enough machinery to prove Proposition 3.3.

Proof of Proposition 3.3. Set

$$d = \lceil \sup_{\mu \in (0,1]} f(\mu, \epsilon, 0) \rceil = (2 + o(1)) \ln(1/\epsilon) \qquad \text{as } \epsilon \to 0$$

as in Lemma 3.2. Next, consider a sample G from $\mathcal{G}(n, \epsilon, d)$ where every left vertex set S with $|S| \geq s+1$ has at least $|S|$ neighbors. Construct a new graph G' by adding s new right vertices and ϵn new left vertices to G, and add edges between every new vertex on each side and every vertex on the other side. Then G' is a bipartite graph with $(1+\epsilon)n$ vertices on each side, and every set S of left vertices has at least $|S|$ neighbors. Hall's theorem then implies that G' has a perfect matching M. Let M' be the subset of M obtained by removing every edge in M incident on at least one vertex in G' that is not present in G. Then $|M| \geq (1+\epsilon)n - \epsilon n - s = n - s$, and M is a matching in G. Applying Lemma 3.6 with $k^* = s+1$ completes the proof. □

The remainder of this section is now devoted to the proof of Theorem 3.2, which we do in essentially the same way as Fotakis et al. [4] prove Theorem 3.1. Consider a sample G from $\mathcal{G}(n, \epsilon, d)$ and let M be a matching in G. Let G'_M denote the graph obtained by reversing the direction of every edge in G_M. Let Y_0 denote the set of right vertices in G not matched in M. For $i \geq 1$, let Y_i denote the set of vertices not in Y_0 that are reachable from Y_0 in G'_M along a path of length at most $2i$.

We now create a new graph G''_M from G'_M in the following way. First, G''_M contains every vertex and edge in G'_M. Then, for each left vertex u that is unmatched by M, we add a new directed edge from u to a distinct vertex in Y_0; this is possible since the number of right vertices is greater than the number of left vertices. Let $Y'_0 \subseteq Y_0$ denote the set of right vertices in G''_M with no incoming edges, and for $i \geq 1$, let Y'_i denote the set of right vertices not in Y'_0 reachable from Y'_0 in G''_M along a path of length at most $2i$. Since every edge in G''_M that is not in G'_M is directed into Y_0, we have $(Y'_0 \cup Y'_i) \subseteq (Y_0 \cup Y_i)$. In particular, $|Y_0 \cup Y_i| \geq |Y'_0 \cup Y'_i|$.

Lemma 3.9. *Suppose that $\epsilon \in (0, 3/8)$ and that G satisfies the expansion property in Lemma 3.8. Then for $\lambda_1 = \lceil \log_{4/3}(1/2\epsilon) \rceil$, we have $|Y'_0 \cup Y'_{\lambda_1}| \geq n(\epsilon + 1/2)$.*

Proof. Since there are n left vertices and $(1+\epsilon)n$ right vertices, and every left vertex has an outgoing edge to a distinct right vertex in G''_M, we have $|Y'_0| = \epsilon n$. Now consider any $i \geq 1$, and let Y''_i be any subset of Y'_i with $|Y''_i| \leq 3n/8$. By the expansion property of G, the set Y''_i has at least $4|Y''_i|/3$ neighbors in G. Since every vertex in Y''_i has at most one incoming edge, at least $|Y''_i|/3$ of these neighbors are also neighbors in G''_M. Each of those $|Y''_i|/3$ neighbors has exactly one outgoing edge, and the right endpoints of those edges are distinct and not in Y''_i or Y'_0 (recall that Y'_0 has no incoming edges in G''_M). Thus, $|Y'_{i+1}| \geq |Y''_i| + |Y''_i|/3 = 4|Y''_i|/3$. Now, if $|Y'_i| \leq 3n/8$, we set $Y''_i = Y'_i$, and if $3n/8 < |Y'_i| \leq (4/3)(3n/8) = n/2$, we let Y''_i be an arbitrary subset of Y'_i of size $3n/8$. It follows that $|Y'_\lambda| \geq n/2$ for some $\lambda \leq 1 + \log_{4/3} \frac{n/2}{|Y'_1|}$. To lower bound $|Y'_1|$, we recall that $|Y'_0| = \epsilon n$, so $\epsilon n \leq |Y'| \leq 3n/8$. By the expansion property of G, the set Y'_0 has at least $4|Y'_0|/3$ neighbors in G. Since Y'_0 has no incoming edges in G''_M, every left vertex has one outgoing edge, and no right vertex has more than one incoming edge, we have $|Y'_1| \geq 4|Y'_0|/3 = (4/3)\epsilon n$. It follows that $|Y'_{\lambda_1}| \geq n/2$. Since $|Y'_0| = \epsilon n$ and $|Y'_0 \cap Y'_{\lambda_1}| = 0$ (by definition of the Y'_i's), we now have $|Y'_0 \cup Y'_{\lambda_1}| \geq n(\epsilon + 1/2)$. □

Lemma 3.10. *Suppose that G satisfies the expansion property in Lemma 3.7 with $\gamma = 1$ and $\delta = 2/3$ and some $1 \leq k^* \leq n/3$; that is, every set of left vertices of size at least k^* but no more than $n/3$ expands by a factor of at least two. Let Y be any set of right vertices with $|Y| \geq (\epsilon + 1/2)n$. Then the number of neighbors of Y in G is at least*

$$n - \max\left(k^* - 1, \frac{(1+\epsilon)n - |Y|}{2}\right).$$

Proof. Let X be the set of left vertices not adjacent to Y. Then $|X| < n/3$, since if $|X| \geq n/3$, it has at least $2n/3$ neighbors, none of which are in Y. In that case, $2n/3 \leq (1+\epsilon)n - |Y|$, contradicting the fact that $|Y| \geq (\epsilon + 1/2)n$. Now if $|X| < k^*$, then we are done. Finally, if $k^* \leq |X| < n/3$, then X has at least $2|X|$ neighbors, none of which are in Y, so $2|X| \leq (1+\epsilon)n - |Y|$. The number of neighbors of Y is then

$$n - |X| \geq n - \frac{(1+\epsilon)n - |Y|}{2}.$$

□

Lemma 3.11. *For $i \geq 0$, let $Z_i = Y'_0 \cup Y'_{\lambda_1+i}$. Suppose that $|Z_0| \geq (\epsilon + 1/2)n$ and that G satisfies the expansion property in Lemma 3.10 for some $k^* \geq 1$. Then for $\lambda_2 = \left\lceil \log \frac{1}{2(\epsilon+(k^*-1)/n)} \right\rceil$, we have that $|Z_{\lambda_2}| \geq (1+\epsilon)n - k^* + 1$.*

Proof. We use induction on $i \geq 0$ to show that

$$|Z_i| \geq \min((1+\epsilon)n - k^* + 1, (1 + \epsilon - 2^{-(i+1)})n).$$

For $i = 0$, we have $|Z_0| \geq (\epsilon + 1/2)n = (1 + \epsilon - 2^{0+1})n$. Now suppose that $i \geq 0$ and that $|Z_i| \geq (1 + \epsilon - 2^{-(i+1)})n$. Since $|Z_i| \geq (\epsilon + 1/2)n$, the expansion property of G implies that Z_i has at least $\min(n - k^* + 1, ((1-\epsilon)n + |Z_i|)/2)$ neighbors. Each of these neighbors has exactly one outgoing edge in G''_M, and the endpoints of these edges are distinct since no right vertex has more than one incoming edge. Furthermore, the ϵn vertices in Y'_0 have no incoming edges. Thus,

$$|Z_{i+1}| \geq \min\left(n - k^* + 1, \frac{(1-\epsilon)n + |Z_i|}{2}\right) + \epsilon n \geq \min((1+\epsilon)n - k^* + 1, (1 + \epsilon - 2^{-(i+2)})n).$$

It follows that $|Z_{\lambda_2}| \geq (1+\epsilon)n - k^* + 1$, as required. □

Lemma 3.12. *Fix $\epsilon > 0$ and $1 \leq k^* \leq n/3$, and let $t = \lambda_1 + \lambda_2$. Let G denote the sample from $\mathcal{G}(n, \epsilon, d)$ representing the hash functions, and suppose that G satisfies the expansion properties in Lemma 3.8 and in Lemma 3.10 with $\gamma = 1$ and $\delta = 2/3$ and k^*, and recall that S is the maximum size of the stash during the insertion of the n items. Then $S < k^*$.*

Proof. For the sake of contradiction, assume that at some point during insertion of the n items, there is some item x_{k^*} for which the original insertion algorithm would yield a failure, and there are items $x_1, \ldots, x_{k^*-1}$ already in the stash, none of which could be inserted using the original insertion algorithm without generating a failure. Let M denote the matching in G representing the placement of items in the hash table when the failure occurs. (Note that previously in our analysis, M was an arbitrary matching, so we may apply our previous notation and results.)

Let X be the set of k^* left vertices corresponding to $x_1, \ldots, x_{k^*}$, and let Y be the neighbors of X. By the expansion property of G, we have that $|Y| \geq 2k^*$. Let $Z = Y_0 \cup Y_t$ and $Z' = Y'_0 \cup Y'_t$. By Lemmas 3.9 and 3.11, we have $|Z'| \geq (1+\epsilon)n - k^* + 1$. We have already shown (just before Lemma 3.9) that $Z' \subseteq Z$, so $|Z| \geq (1+\epsilon)n - k^* + 1$. Thus, there is some $y \in Y \cap Z$. Since $y \in Z$, there is a path from y to Y_0 of length at most $2t$ in G_M. Since y is a neighbor of X, there must be a path from some $x \in X$ to Y_0 in G_M with length at most $2t + 1$. This implies that attempting to insert x would result in it being successfully placed in the hash table, yielding a contradiction. □

We can now prove Theorem 3.2.

Proof of Theorem 3.2. Set

$$d = \left\lceil \max\left(\sup_{\mu \in (0,1/3]} f(\mu, \epsilon, 1), \quad 5 + 3\ln(1/\epsilon) \right) \right\rceil = 3\ln(1/\epsilon) + O(1) \qquad \text{as } \epsilon \to 0,$$

as in Lemma 3.2. Next, let G denote the sample from $\mathcal{G}(n, \epsilon, d)$ corresponding to the hash functions. By Lemma 3.12, we can only have $S \geq s$ if G lacks either the expansion property of Lemma 3.8 or the expansion property of Lemma 3.10 with $\gamma = 1$, $\delta = 2/3$, and $k^* = s$. By Lemmas 3.7, 3.8, and 3.10, both of these events occur with probability $O(n^{1-cs})$. A union bound now completes the proof. □

4 A Variant with Multiple Items per Bucket

The last scheme we consider in this work is the cuckoo hashing variant proposed by Dietzfelbinger and Weidling [2]. Here we attempt to insert n items into a table with $m = (1+\epsilon)n/d$ buckets using two hash functions h_1 and h_2, for some constants $\epsilon, d > 0$. Each bucket can hold at most d items. (Note that here, following the notation of [2], d is the capacity of a bucket, not the number of hash functions as in Section 3; here there are two hash functions.) As before, we assume that the hash functions are independent and fully random.

We think of hash functions as defining a multi-graph G with m vertices, representing the buckets. Each item x is encoded in G by the edge $(h_1(x), h_2(x))$. We think of a partial placement of the items into the hash locations as directed version of a subgraph G. For some allocation A of the items to the buckets, we let G_A denote the graph obtained by deleting edges from G not corresponding to some $x \in A$, and then orienting each edge towards the bucket containing the corresponding item. Since each bucket can store at most d items, the in-degree of every vertex in G_A is bounded by d.

To perform an insertion a new item x when the current allocation is determined by A, we do a breadth-first search in G_A starting from $\{h_1(x), h_2(x)\}$ in search of a vertex with in-degree less than d. If we find such a vertex v, then we move the items in the table accordingly to simulate adding x to A and reorienting the edges appropriately. If there is no such vertex v, then we resample the hash functions and attempt to reinsert all of the items in the table and x.

The three main results of [2] are the following.

Proposition 4.1. *For any $\epsilon > 0$ and $d \geq 1 + \frac{\ln(1/\epsilon)}{1-\ln 2}$, the probability that it is not possible to orient the edges of G so that every vertex has in-degree at most d is $O(n^{1-d})$.*

Proposition 4.2. *For sufficiently small $\epsilon > 0$, if $6 + 5\ln d + d < \frac{\ln(1/\epsilon)}{1-\ln 2}$, then the probability that it is possible to orient the edges of G so that every vertex has in-degree at most d is $2^{-\Omega(n)}$.*

Theorem 4.1. *For $\epsilon \in (0, 0.1]$ and $d \geq 15.8 \cdot \ln(1/\epsilon)$ and sufficiently large n, there is some constant c such that the expected time to insert each of the n items into the table is at most $(1/\epsilon)^{c \log d}$. Furthermore, the probability that some insertion operation requires a rehash is $O(n^{1-d})$.*

The significance of these results is similar to the ones for generalized cuckoo hashing discussed in Section 3. Proposition 4.1 is a feasibility result, telling us that if d is sufficiently large with respect to ϵ, then with high probability, it is possible to place the items into the buckets while respecting the hash functions. Proposition 4.2 tells us that the lower bound on d from Proposition 4.1 is essentially

tight. Finally, Theorem 4.1 is the most significant result, giving us performance guarantees for the proposed insertion algorithm. As an aside, we note that the high probability bound in Theorem 4.1 follows immediately from Proposition 4.1. While this bound is not nearly as impressive as the bound on the expected insertion time, our stash technique only improves the high probability bound. (Note that this is the case for the stash technique in all of our examples; it only drives down the probability of exceptionally bad events, and never decreases the asymptotic average cost of an insertion operation.) Still, our results are another nice demonstration of how the stash technique can be easily applied to an already existing hashing scheme, and so we include them as further evidence of the technique's versatility.

We are now ready to describe our results. First, we generalize Propositions 4.1 and Propositions 4.2 to the case where a constant number of items can be stored in a stash. The new results strongly suggest that the stash technique can be usefully applied to this hashing scheme.

Proposition 4.3. *For any $\epsilon > 0$, $d \geq 1 + \frac{\ln(1/\epsilon)}{1-\ln 2}$, and $s \geq 0$, the probability that there is a subset of s edges in G whose removal allows us to orient the remaining edges of G so that every vertex has in-degree at most d is $1 - O(n^{(s+1)(1-d)})$.*

Proof. Proposition 4.1 establishes the case where $s = 0$, and so we assume that $s \geq 1$. The proof is now obtained by some simple modifications to the proof of Proposition 4.1 in [2, Section 3.2]. First, we change the definition of F to $F = \mathbf{Pr}(\exists X \subseteq S, |X| \geq s+1 : \Gamma(X) \leq |X|/d)$. The inequality [2, (10)] then becomes $F \leq \sum_{s+1 \leq j \leq m/(1+\epsilon)} F(j)$. Case 1 is now no longer necessary, and the proofs of Case 3 and Case 4 can be left as they are. Case 2 becomes $s+1 \leq j \leq m/(2e^4)$. The inequality [2, (18)] then becomes

$$\sum_{s+1 \leq j \leq m/(2e^4)} F(j) = O(f(s+1,0)) = O\left(\left(\left(\frac{s+1}{m}\right)^{d-1} e^{d+1}\right)^{s+1}\right) = O(n^{(s+1)(1-d)}),$$

completing the proof. □

Proposition 4.4. *For sufficiently small $\epsilon > 0$ and any constant $s \geq 0$, if $6 + 5\ln d + d < \frac{\ln(1/\epsilon)}{1-\ln 2}$, then the probability that it is possible to remove s edges from G and orient the remaining edges so that every vertex has in-degree at most d is $2^{-\Omega(n)}$.*

Proof. We proceed by directly modifying the proof of [2, Proposition 3.1]. We make no modification the proof through [2, (7)]. From that point on, we alter the text so that it reads as follows:

> If this expected value is larger by a constant factor than $\epsilon n + s$ (e.g., $mp_{d-1} \geq 1.05(\epsilon n + s)$), then with probability $1 - 2^{-\Omega(n)}$ more than $\epsilon n + s$ buckets are hit by exactly $d-1$ hash values. (This follows by applying standard tail inequalities like the Azuma-Hoeffding inequality.) These buckets can not be full in any allocation of $n - s$ items to the buckets, implying that it is impossible to place $n - s$ items into the table while respecting the hash functions. In this case, the removal of any s edges from G does not allow us to orient the remaining edges so that every vertex has in-degree at most d. By (6), a sufficient condition for this situation is:
>
> $$\frac{(2/e)^d}{5.2\sqrt{d}} \cdot m > 1.05(\epsilon n + s),$$

which, for sufficiently large n, is satisfied if

$$\frac{(2/e)^d}{5.2\sqrt{d}} \cdot m > 1.1\epsilon n.$$

The last relationship above is exactly the same as [2, (8)]. Applying the arguments from [2] from that point on completes the proof. □

We now describe how to modify the insertion procedure to allow for the presence of a stash. The idea is basically the same as the modification of the generalized cuckoo hashing insertion algorithm analyzed in Section 3. We simply use the original insertion algorithm, except that if, during the insertion of some item x, that algorithm would have us perform a rehash of all the items in the table. In that case, we simply add x to the stash.

Note that our modified insertion algorithm never causes the items in the table to be rehashed. However, if we were to work with the variant where the stash has bounded size, we would perform a rehash of all of the items when the stash overflows. Thus, while we will not show a result concerning the expected time to insert an item, the variant that allows rehashings is easily seen to give the same asymptotic time bound for the insertion of an item as in Theorem 4.1. For this reason, we focus on proving a counterpart to the high probability result in Theorem 4.1 for the first modification, where the stash has unbounded size and no rehashings are ever performed. (Once again, the stash technique only allows us to decrease the probability of exceptionally bad events; it does not allow us to reduce the asymptotic average cost of an insertion operation.) Our main result is the following theorem.

Theorem 4.2. *Let S denote the size of the stash after all n items have been inserted. For sufficiently small $\epsilon > 0$ and $d \geq 15.8 \cdot \ln(1/\epsilon)$ and any integer $s \geq 2$, we have $\mathbf{Pr}(S \geq s) = O(n^{sd(\Delta-1)})$, for $\Delta = d^{-1/3} + d^{-1}$. For $s = 1$, we have $\mathbf{Pr}(S \geq s) = O(n^{1-d})$.*

The rest of this section is devoted to the proof of Theorem 4.2, which is essentially just some simple modifications to the proof of Theorem 4.1 given in [2]. As in Section 3, the main idea of the proof is that G satisfies certain expansion properties with high probability, such that if G satisfies these expansion properties, we are guaranteed to have $S < s$. For brevity, we use the notation of [2] in what follows, and do not bother to redefine it.

Lemma 4.1. *For every $s \geq 1$, the probability that there is a set X of at least s edges that does not hit at least $|X|$ buckets is $O(n^{s(1-d)})$.*

Proof. This is simply the bound on F shown in the proof of Proposition 4.3 for $s - 1$. □

Lemma 4.2. *Let $\gamma = 4/(e^4 d^3)$. For sufficiently large (constant) d, for every $s \geq 1$, the probability that there is a set X of edges with $sd \leq |X| \leq \gamma dm$ that does not hit at least $\Delta|X|$ different bins is $O(n^{sd(\Delta-1)})$.*

Proof. The proof is essentially the same as for [2, Lemma 6]. We simply bound the probability of

interest by

$$\begin{aligned}\sum_{sd\le j\le\gamma dm}\binom{n}{j}\binom{m}{\Delta j}\left(\frac{\Delta j}{m}\right)^{2j} &\le \sum_{sd\le j\le\gamma dm}\left(\frac{ne}{j}\right)^{j}\left(\frac{me}{\Delta j}\right)^{j}\left(\frac{\Delta j}{m}\right)^{2j}\\ &= \sum_{sd\le j\le\gamma dm}\left(\left(\frac{j}{n}\right)^{1-\Delta}\right)^{j}\left(\frac{e^{1+\Delta}\Delta^{2-\Delta}(1+\epsilon)^{2-\Delta}}{d^{2-\Delta}}\right)^{j}\\ &= O(n^{sd(\Delta-1)}).\end{aligned}$$

The first step follows by a standard bound on binomial coefficients. The second step is obvious, and the third step follows from the fact that for sufficiently large d, the terms in the second step decrease geometrically. □

Lemma 4.3. *Suppose that $m - |Y_{k^*+\ell^*}| \le \gamma m$ and the hash functions h_1 and h_2 satisfy the conclusion of Lemma 4.2 for $s \ge 1$. Let $a_j = m - |Y_{k^*+\ell^*+j}|$ for $j \ge 0$. Then for any $j \ge 0$, we have $a_j \le \max(s-1, d^{-2/3}a_{j-1})$.*

Proof. Fix some $j \ge 1$. We assume that $a_j \ge s$, since otherwise the result is trivial. By definition, all a_j buckets in $[m]-Y_{k^*+\ell^*+j}$ are full, and so there are da_j edges with destinations in $[m]-Y_{k^*+\ell^*+j}$. Invoking the conclusion of Lemma 4.2 tells us that these edges touch at least Δda_j vertices, only a_j of which are in $[m] - Y_{k^*+\ell^*+j}$. Since there can be no edges from $[m] - Y_{k^*+\ell^*+j}$ to $Y_{k^*+\ell^*+(j-1)}$, the origins of these edges hit at least $\Delta da_j - a_j = d^{2/3}a_j$ distinct vertices in $[m] - Y_{k^*+\ell^*+(j-1)}$, implying that $a_{j-1} \ge d^{2/3}a_j$. □

We are now ready to prove Theorem 4.2.

Proof of Theorem 4.2. We focus on the case $s \ge 2$, since the case $s = 1$ follows immediately from Theorem 4.1. First, we note that with probability $1 - O(n^{sd(\Delta-1)})$, the graph G satisfies the conclusion of Lemma 4.1, as well as the expansion properties of [2, Lemmas 2 and 4] and Lemma 4.2. In this case, G satisfies the expansion properties of [2, Lemmas 3 and 5] and Lemma 4.3. To complete the proof, we show that this implies $S < s$.

For the sake of contradiction, suppose that the stash contains a set of items $X = \{x_1, \ldots, x_s\}$ after all n items have been inserted. By the expansion property of Lemma 4.3, for a sufficiently large $j^* \ge 0$, the set $[m] - Y_{k^*+\ell^*+j^*}$ has fewer than s items. By the conclusion of Lemma 4.1, the items in X hit at least s buckets. Thus, there is some $x \in X$ such that if we were to attempt to insert x again, it would be successfully placed in the table. But x was not successfully placed in the table when it was originally inserted, and it is easy to see that inserting more items into the table cannot create the possibility for x to be successfully inserted after that point. Thus, we have derived a contradiction, completing the proof. □

5 Some Simple Experiments

In order to demonstrate the potential importance of our results in practical settings, we present some simple experiments. We emphasize that these experiments are not intended as a rigorous empirical study; they are intended only to be suggestive of the practical relevance of the general stash technique. First, we consider using a cuckoo hash table with $d = 2$ choices, consisting of

Stash Size	Standard	Modified
0	992812	992919
1	6834	6755
2	338	307
3	17	15
4	1	2

(a) 1000 items

Stash Size	Standard	Modified
0	9989861	9989571
1	10040	10350
2	97	78
3	2	1
4	0	0

(b) 10000 items

Table 1: For $d = 2$, failures measured over 10^6 trials for 1000 items, requiring a maximum stash size of four (a), and failures measured over 10^7 trials, requiring a maximum stash size of three (b).

two sub-tables of size 1200. We insert 1000 items, allowing up to 100 evictions before declaring a failure and putting some item into the stash. In this experiment we allow the stash to hold as many items as needed; the number of failures gives the size the stash would need to be to avoid rehashing or a similar failure mode. In our experiments, we use the standard Java pseudorandom number generator to obtain hash values. We consider both standard cuckoo hashing, where after 100 evictions the last item evicted is moved to the stash, and the slightly modified version considered in Section 2, where if an item is not placed after 100 evictions, we reverse the insertion operation and redo it, this time looking for a "bad edge" in the cuckoo graph to place in the stash. Recall that this removal process was important to our analysis.

The results from one million trials are presented in Table 1a. As expected, in most cases, in fact over 99% of the time, no stash is needed. The simple expedient of including a stash that can hold just 4 items, however, appears to reduce the probability for a need to rehash to below 10^{-6}. A slightly larger stash would be sufficient for most industrial strength applications, requiring much less memory than expanding the hash table to achieve similar failure rates. It is worth noting that there appears to be little difference between standard hashing and the modified version. It would be useful in the future to prove this formally.

We show similar results for placing 10000 items using $d = 2$ choices with two sub-tables of size 12000 in Table 1b. Here we use 10^7 trials in order to obtain a meaningful comparison. The overall message is the same: a very small stash greatly reduces the probability that some item cannot be placed effectively.

We also conduct some simple experiments for the case $d = 3$. Here, rather than considering the breadth-first search extension of cuckoo hashing analyzed in Section 3, we examine the more practical *random walk* variant introduced in [4]. We use d equally sized tables, which we think of as arranged from left to right, with one hash function per table. To insert an item, we check whether any of its hash locations are unoccupied, and in that case, we place it in its leftmost unoccupied hash location. Otherwise, we place it in a randomly chosen hash location and evict the item x in that place. Then we check if any of x's hash locations are empty, and if so, we place it in its leftmost unoccupied hash location. Otherwise, we place x in a randomly chosen hash location that is different from the hash location from which it was just evicted, evicting the item y in that place. We continue in this way until we successfully place an item or we perform some prespecified number of evictions, in which case we place the last item evicted into the stash.

We also consider a variant corresponding to a random walk version of the insertion procedure analyzed in Section 3. Here, just before inserting an item x into a stash consisting of items $x_1, \ldots, x_s$,

Stash Size	Standard	Modified
0	998452	998490
1	1537	1510
2	11	0

(a) 1000 items

Stash Size	Standard	Modified
0	9964148	9964109
1	35769	35891
2	83	0

(b) 10000 items

Table 2: For $d = 3$, failures measured over 10^6 trials for 1000 items (a), and failures measured over 10^7 trials (b).

we do the following. For each $i = 1, \ldots, s$, we remove x_i from the stash and insert it into the table following the standard random walk insertion procedure. If this procedure terminates within the prespecified number of evictions, we place x at position i in the stash, and conclude the insertion operation. Otherwise, we place the last item y evicted during this reinsertion of x_i at position i in the stash. Finally, if none of the reinsertions of $x_1, \ldots, x_s$ successfully completes within the prespecified number of evictions, then we place x at position $s + 1$ in the stash.

Specifically, we consider the case where $d = 3$, there are 1000 items, each of the three tables has size 400, and the maximum number of evictions is 100. As before, we perform 10^6 trials. Similarly, for 10000 items, we consider the same experiment with three tables of size 4000, and perform 10^7 trials. The results are displayed in Table 2, and are analogous to those in Table 1.

6 Conclusion

We have shown how to greatly improve the failure probability bounds for a large class of cuckoo hashing variants by using only a constant amount of additional space. Furthermore, our proof techniques naturally extend the analysis of the original schemes in a straightforward way, strongly suggesting that our techniques will continue to be broadly applicable for future hashing schemes. Finally, we have also presented some simple experiments demonstrating that our improvements have real practical potential.

There remain several open questions. As a technical question, it would be useful to extend our analysis to work with the original cuckoo hashing variants, in place of the modified variants we have described. More importantly, the analysis of random-walk variants when $d > 2$, in place of breadth-first-search variants, remains open both with and without a stash. A major open question is proving the above bounds for *explicit* hash families that can be represented, sampled, and evaluated efficiently. Such explicit constructions are provided for standard cuckoo hashing in [11] and [3]. It would be interesting to improve upon those constructions and extend them to the case of a stash.

Acknowledgment

The authors are grateful to Thomas Holenstein for useful discussions.

References

[1] L. Devroye and P. Morin. Cuckoo Hashing: Further Analysis. *Information Processing Letters*, 86(4):215-219, 2003.

[2] M. Dietzfelbinger and C. Weidling. Balanced Allocation and Dictionaries with Tightly Packed Constant Size Bins. *Theoretical Computer Science*, 380(1-2):47-68, 2007.

[3] M. Dietzfelbinger and P. Woelfel. Almost Random Graphs with Simple Hash Functions. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, pp. 629-638, 2003.

[4] D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space Efficient Hash Tables With Worst Case Constant Access Time. *Theory of Computing Systems*, 38(2):229-248, 2005.

[5] A. Kirsch and M. Mitzenmacher. The Power of One Move: Hashing Schemes for Hardware. To appear in *Proceedings of the 27th IEEE International Conference on Computer Communications* (INFOCOM), 2008. Temporary version available at `http://www.eecs.harvard.edu/~kirsch/pubs/`.

[6] A. Kirsch and M. Mitzenmacher. Using a Queue to De-amortize Cuckoo Hashing in Hardware. In *Proceedings of the Forty-Fifth Annual Allerton Conference on Communication, Control, and Computing*, 2007.

[7] A. Kirsch, M. Mitzenmacher, and U. Wieder. More Robust Hashing: Cuckoo Hashing with a Stash. To appear in *Proceedings of the 16th Annual European Symposium on Algorithms* (ESA), 2008. Temporary version available at `http://www.eecs.harvard.edu/~kirsch/pubs/`.

[8] R. Kutzelnigg. Bipartite Random Graphs and Cuckoo Hashing. In *Proceedings of the Fourth Colloquium on Mathematics and Computer Science*, 2006.

[9] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[10] M. Naor, G. Segev, and U. Wieder. History Independent Cuckoo Hashing. Manuscript, 2008.

[11] R. Pagh and F. Rodler. Cuckoo Hashing. *Journal of Algorithms*, 51(2):122-144, 2004.