

Explicit and Efficient Hash Families Suffice for Cuckoo Hashing with a Stash

Martin Aumüller¹, Martin Dietzfelbinger^{1,*}, and Philipp Woelfel^{2,**}

¹ Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau,
98694 Ilmenau, Germany

martin.aumueller@tu-ilmenau.de, martin.dietzfelbinger@tu-ilmenau.de

² Department of Computer Science, University of Calgary,
Calgary, Alberta T2N 1N4, Canada
woelfel@cpsc.ucalgary.ca

Abstract. It is shown that for cuckoo hashing with a stash as proposed by Kirsch, Mitzenmacher, and Wieder (2008) families of very simple hash functions can be used, maintaining the favorable performance guarantees: with stash size s the probability of a rehash is $O(1/n^{s+1})$, and the evaluation time is $O(s)$. Instead of the full randomness needed for the analysis of Kirsch *et al.* and of Kutzelnigg (2010) (resp. $\Theta(\log n)$ -wise independence for standard cuckoo hashing) the new approach even works with 2-wise independent hash families as building blocks. Both construction and analysis build upon the work of Dietzfelbinger and Woelfel (2003). The analysis, which can also be applied to the fully random case, utilizes a graph counting argument and is much simpler than previous proofs. As a byproduct, an algorithm for simulating uniform hashing is obtained. While it requires about twice as much space as the most space efficient solutions, it is attractive because of its simple and direct structure.

1 Introduction

Cuckoo hashing as proposed by Pagh and Rodler [17] is a popular implementation of a dictionary with guaranteed constant lookup time. To store a set S of n keys from a universe U (i.e., a finite set), cuckoo hashing utilizes two hash functions, $h_1, h_2 : U \rightarrow [m]$, where $m = (1 + \varepsilon)n$, $\varepsilon > 0$. Each key $x \in S$ is stored in one of two hash tables of size m ; either in the first table at location $h_1(x)$ or in the second one at location $h_2(x)$. The pair h_1, h_2 might not be suitable to accommodate S in these two tables. In this case, a *rehash* operation is necessary, which chooses a new pair h_1, h_2 and inserts all keys anew.

In their ESA 2008 paper [11], Kirsch, Mitzenmacher, and Wieder deplored the order of magnitude of the probability of a rehash, which is as large as $\Theta(1/n)$. They proposed adding a *stash*, an additional segment of storage that can hold up to s keys for some (constant) parameter s , and showed that this change

* Research supported by DFG grant DI 412/10-2.

** Research supported by a Discovery Grant from the National Sciences and Research Council of Canada (NSERC).

reduces the rehash probability to $\Theta(1/n^{s+1})$. However, the analysis of Kirsch *et al.* requires the hash functions to be fully random. In the journal version [12] Kirsch *et al.* posed “proving the above bounds for explicit hash families that can be represented, sampled, and evaluated efficiently” as an open problem.

Our contribution. In this paper we generalize a hash family construction proposed by Dietzfelbinger and Woelfel [9] and show that the resulting hash functions have random properties strong enough to preserve the qualities of cuckoo hashing with a stash. The proof involves a new and simpler analysis of this hashing scheme, which also works in the fully random case. The hash functions we propose have a very simple structure: they combine functions from $O(1)$ -wise independent families³ with a few tables of size $n^{1-\Theta(1)}$ with random entries from $[m] = \{0, \dots, m-1\}$. An attractive version of the construction for stash capacity s has the following performance characteristics: the description of a hash function pair (h_1, h_2) consists of a table with \sqrt{n} entries from $[m]^2$ and $2s+6$ functions from 2-wise independent classes. To evaluate $h_1(x)$ and $h_2(x)$ for $x \in U$, we must evaluate these $2s+6$ functions, read $2s+4$ table entries, and carry out $4s+8$ additions modulo m . Our main result implies for these hash functions and for any set $S \subseteq U$ of n keys that with probability $1 - O(1/n^{s+1})$ S can be accommodated according to the cuckoo hashing rules.

In addition, we present a simple data structure for simulating a uniform hash function on S with range R , using our hash class and essentially a table with $2(1+\varepsilon)n$ random elements from R .

Cuckoo hashing with a stash and weak hash functions. In [12,14] it was noticed that for the analysis of cuckoo hashing with a stash of size s the properties of the so-called *cuckoo graph* $G(S, h_1, h_2)$ are central. Assume a set S and hash functions h_1 and h_2 with range $[m]$ are given. The associated cuckoo graph $G(S, h_1, h_2)$ is the bipartite multigraph whose two node sets are copies of $[m]$ and whose edge set contains the n pairs $(h_1(x), h_2(x))$, for $x \in S$. It is known that a single parameter of $G = G(S, h_1, h_2)$ determines whether a stash of size s is sufficient to store S using (h_1, h_2) , namely the *excess* $\text{ex}(G)$, which is defined as the minimum number of edges one has to remove from G so that all connected components of the remaining graph are acyclic or unicyclic.

Lemma 1 ([12]). *The keys from S can be stored in the two tables and a stash of size s using (h_1, h_2) if and only if $\text{ex}(G(S, h_1, h_2)) \leq s$.*

For the convenience of the reader, a proof is given in Appendix A, along with a discussion of insertion procedures, which is omitted in the main text.

Kirsch *et al.* [12] showed that with probability $1 - O(1/n^{s+1})$ a random bipartite graph with $2m = 2(1+\varepsilon)n$ nodes and n edges has excess at most s . Their proof uses sophisticated tools such as Poissonization and Markov chain coupling. This result generalizes the analysis of standard cuckoo hashing [17] with no stash, in which the rehash probability is $\Theta(1/n)$. Kutzelnigg [14] refined the analysis

³ κ -wise independent families of hash functions are defined in Section 2.

of [12] in order to determine the constant factor in the asymptotic bound of the rehash probability. His proof uses generating functions and differential recurrence equations. Both approaches inherently require that the hash functions h_1 and h_2 used in the algorithm are fully random.

Recently, Pătraşcu and Thorup [18] showed that simple tabulation hash functions are sufficient for running cuckoo hashing, with a rehash probability of $\Theta(1/n^{1/3})$, which is tight. Unfortunately, for these hash functions the rehash probability cannot be improved by using a stash.

Our main contribution is a new analysis that shows that explicit and efficient hash families are sufficient to obtain the $O(1/n^{s+1})$ bound on the rehash probability. We build upon the work of Dietzfelbinger and Woelfel [9]. For standard cuckoo hashing, they proposed hash functions of the form $h_i(x) = (f_i(x) + z^{(i)}[g(x)]) \bmod m$, for $x \in U$, for $i \in \{1, 2\}$, where f_i and g are from $2k$ -wise independent classes with range $[m]$ and $[\ell]$, resp., and $z^{(1)}, z^{(2)} \in [m]^\ell$ are random vectors. They showed that with such hash functions the rehash probability is $O(1/n + n/\ell^k)$. Their proof has parts (i) and (ii). Part (i) already appeared in [3] and [17]: The rehash probability is bounded by the sum, taken over all minimal excess-1 graphs H of different sizes and all subsets T of S , of the probability that $G(T, h_1, h_2)$ is isomorphic to H . In Sect. 5 of this paper we demonstrate that for h_1 and h_2 fully random a similar counting approach also works for minimal excess- $(s+1)$ graphs, whose presence in $G(S, h_1, h_2)$ determines whether a rehash is needed when a stash of size s is used. As in [17], this analysis also works for $O((s+1) \log n)$ -wise independent families.

Part (ii) of the analysis in [9] is a little more subtle. It shows that for each key set S of size n there is a part B_S^{conn} of the probability space given by (h_1, h_2) such that $\Pr(B_S^{\text{conn}}) = O(n/\ell^k)$ and in $\overline{B}_S^{\text{conn}}$ the hash functions act fully randomly on $T \subseteq S$ as long as $G(T, h_1, h_2)$ is connected. In Sect. 4 we show how this argument can be adapted to the situation with a stash, using subgraphs without leaves in place of the connected subgraphs. Woelfel [23] already demonstrated by applying functions in [9] to balanced allocation that the approach has more general potential to it.

A comment on the “full randomness assumption” and work relating to it seems in order. It is often quoted as an empirical observation that weaker hash functions like κ -wise independent families will behave almost like random functions. Mitzenmacher and Vadhan [15] showed that if the key set S has a certain kind of entropy then 2-wise independent hash functions will behave similar to fully random ones. However, as demonstrated in [7], there are situations where cuckoo hashing fails for a standard 2-wise independent family and even a random set S (which is “too dense” in U). The rather general “split-and-share” approach of [6] makes it possible to justify the full randomness assumption for many situations involving hash functions, including cuckoo hashing and many of its variants. However, for practical application this method is less attractive, since space consumption and failure probability are negatively affected by splitting the key set into “chunks” and treating these separately.

Simulating Uniform Hashing. Consider a universe U of keys and a finite set R . By the term “*simulating uniform hashing for U and R* ” we mean an algorithm that does the following. On input $n \in \mathbb{N}$, a randomized procedure sets up a data structure DS_n that represents a hash function $h: U \rightarrow R$, which can then be evaluated efficiently for keys in U . For each set $S \subseteq U$ of cardinality n there is an event B_S with the property that conditioned on $\overline{B_S}$ the values $h(x)$, $x \in S$, are fully random. The quality of the algorithm is determined by the space needed for DS_n , the evaluation time for h , and the probability of the event B_S . It should be possible to evaluate h in constant time. The amount of entropy required for such an algorithm implies that at least $n \log |R|$ bits are needed to represent DS_n .

Pagh and Pagh [16] proposed a construction with $O(n)$ random words from R , based on Siegel’s functions [19], which have constant, but huge evaluation time. They also gave a general method to reduce the space to $(1 + \varepsilon)n$, at the cost of an evaluation time of $O(1/\varepsilon^2)$. In [9] a linear-space construction with tables of size $O(n)$ was given that contain (descriptions of) $O(1)$ -wise independent hash functions. The construction with the currently asymptotically best performance parameters, $(1 + \varepsilon)n$ words from R and evaluation time $O(\log(1/\varepsilon))$, as given in [6], is based on results of Calkin [1] and the “split-and-share” approach, involving the same disadvantages as mentioned above.

Our construction, to be described in Sect. 6, essentially results from the construction in [16] by replacing Siegel’s functions with functions from our new class. The data structure consists of a hash function pair (h_1, h_2) from our hash class, a $O(1)$ -wise independent hash function with range R , $O(s)$ small tables with entries from R , and two tables of size $m = (1 + \varepsilon)n$ each, filled with random elements from R . The evaluation time of h is $O(s)$, and for $S \subseteq U$, $|S| = n$, the event B_S occurs with probability $O(1/n^{s+1})$. The construction requires roughly twice as much space as the most space-efficient solutions [6,16]. However, it seems to be a good compromise combining simplicity with moderate space consumption.

2 Basics

Let U (the “universe”) be a finite set. A mapping from U to $[r]$ is a *hash function with range $[r]$* . For an integer $\kappa \geq 2$, a set \mathcal{H} of hash functions with range $[r]$ is called a *κ -wise independent hash family* if for arbitrary distinct keys $x_1, \dots, x_\kappa \in U$ and for arbitrary $j_1, \dots, j_\kappa \in [r]$ we have $\Pr_{h \in \mathcal{H}}(h(x_1) = j_1 \wedge \dots \wedge h(x_\kappa) = j_\kappa) = 1/r^\kappa$. The classical κ -wise independent hash family construction is based on polynomials of degree $\kappa - 1$ over a finite field [22]. More efficient hash function evaluation can be achieved with tabulation-based constructions [9,20,21,13]. Throughout this paper, \mathcal{H}_r^κ denotes an arbitrary κ -wise independent hash family with domain U and range $[r]$.

We combine κ -wise independent classes with lookups in tables of size ℓ in order to obtain pairs of hash functions from U to $[m]$:

Definition 1. Let $c \geq 1$ and $\kappa \geq 2$. For integers $m, \ell \geq 1$, and given $f_1, f_2 \in \mathcal{H}_m^\kappa$, $g_1, \dots, g_c \in \mathcal{H}_\ell^\kappa$, and vectors $z_j^{(i)} \in [m]^\ell$, $1 \leq j \leq c$, for $i \in \{1, 2\}$, let $(h_1, h_2) = (h_1, h_2) \langle f_1, f_2, g_1, \dots, g_c, z_1^{(1)}, \dots, z_c^{(1)}, z_1^{(2)}, \dots, z_c^{(2)} \rangle$, where

$$h_i(x) = \left(f_i(x) + \sum_{1 \leq j \leq c} z_j^{(i)} [g_j(x)] \right) \bmod m, \text{ for } x \in U, i \in \{1, 2\}.$$

Let $\mathcal{Z}_{\ell, m}^{\kappa, c}$ be the family of all these pairs (h_1, h_2) of hash functions.

While this is not reflected in the notation, we consider (h_1, h_2) as a structure from which the components g_1, \dots, g_c and $f_i, z_1^{(i)}, \dots, z_c^{(i)}$, $i \in \{1, 2\}$, can be read off again. It is family $\mathcal{Z} = \mathcal{Z}_{\ell, m}^{2k, c}$, for some $k \geq 1$, made into a probability space by the uniform distribution, that we will study in the following. We usually assume that c and k are fixed and that m and ℓ are known.

2.1 Basic Facts

We start with some basic observations concerning the effects of compression properties in the “ g -part” of (h_1, h_2) , extending similar statements in [9].

Definition 2. For $T \subseteq U$, define the random variable d_T , the “deficiency” of (h_1, h_2) with respect to T , by $d_T((h_1, h_2)) = |T| - \max\{k, |g_1(T)|, \dots, |g_c(T)|\}$. (Note: d_T depends only on the g_j -components of (h_1, h_2) .) Further, define

- (i) bad_T as the event that $d_T > k$;
- (ii) good_T as $\overline{\text{bad}_T}$, i. e., the event that $d_T \leq k$;
- (iii) crit_T as the event that $d_T = k$.

Hash function pairs (h_1, h_2) in these events are called “ T -bad”, “ T -good”, and “ T -critical”, resp.

Lemma 2. Assume $k \geq 1$ and $c \geq 1$. For $T \subseteq U$, the following holds:

- (a) $\Pr(\text{bad}_T \cup \text{crit}_T) \leq (|T|^{2k}/\ell^k)^c$.
- (b) Conditioned on good_T (or on crit_T), the pairs $(h_1(x), h_2(x))$, $x \in T$, are distributed uniformly and independently in $[r]^2$.

Proof. (a) Assume $|T| \geq 2k$ (otherwise the events bad_T and crit_T cannot occur). Since g_1, \dots, g_c are independent, it suffices to show that for a function g chosen randomly from \mathcal{H}_ℓ^{2k} we have $\Pr(|T| - |g(T)| \geq k) \leq |T|^{2k}/\ell^k$.

We first argue that if $|T| - |g(T)| \geq k$ then there is a subset T' of T with $|T'| = 2k$ and $|g(T')| \leq k$. Initialize T' as T . Repeat the following as long as $|T'| > 2k$: (i) if there exists a key $x \in T'$ such that $g(x) \neq g(y)$ for all $y \in T' \setminus \{x\}$, remove x from T' ; (ii) otherwise, remove any key. Clearly, this process terminates with $|T'| = 2k$. It also maintains the invariant $|T'| - |g(T')| \geq k$: In case (i) $|T'| - |g(T')|$ remains unchanged. In case (ii) before the key is removed from T' we have $|g(T')| \leq |T'|/2$ and thus $|T'| - |g(T')| \geq |T'|/2 > k$.

Now fix a subset T' of T of size $2k$ that satisfies $|g(T')| \leq k$. The preimages $g^{-1}(u)$, $u \in g(T')$, partition T' into k' classes, $k' \leq k$, such that g is constant on each class. Since g is chosen from a $2k$ -wise independent class, the probability

that g is constant on all classes of a given partition of T' into classes $C_1, \dots, C_{k'}$, with $k' \leq k$, is exactly $\ell^{-(2k-k')} \leq \ell^{-k}$.

Finally, we bound $\Pr(|g(T)| \leq |T| - k)$. There are $\binom{|T|}{2k}$ subsets T' of T of size $2k$. Every partition of such a set T' into $k' \leq k$ classes can be represented by a permutation of T' with k' cycles, where each cycle contains the elements from one class. Hence, there are at most $(2k)!$ such partitions. This yields:

$$\Pr(|T| - |g(T)| \geq k) \leq \binom{|T|}{2k} \cdot (2k)! \cdot \frac{1}{\ell^k} \leq \frac{|T|^{2k}}{\ell^k}. \quad (1)$$

(b) If $|T| \leq 2k$, then h_1 and h_2 are fully random on T simply because f_1 and f_2 are $2k$ -wise independent. So suppose $|T| > 2k$. Fix an arbitrary g -part of (h_1, h_2) so that good_T occurs, i.e., $\max\{|g_1(T)|, \dots, |g_c(T)|\} \geq |T| - k$. Let $j_0 \in \{1, \dots, c\}$ be such that $|g_{j_0}(T)| \geq |T| - k$. Arbitrarily fix all values in the tables $z_j^{(i)}$ with $j \neq j_0$ and $i \in \{1, 2\}$. Let T^* be the set of keys in T colliding with other keys in T under g_{j_0} . Then $|T^*| \leq 2k$. Choose the values $z_{j_0}^{(i)}[g_{j_0}(x)]$ for all $x \in T^*$ and $i \in \{1, 2\}$ at random. Furthermore, choose f_1 and f_2 at random from the $2k$ -wise independent family \mathcal{H}_r^{2k} . This determines $h_1(x)$ and $h_2(x)$, $x \in T^*$, as fully random values. Furthermore, the function g_{j_0} maps the keys $x \in T - T^*$ to distinct entries of the vectors $z_{j_0}^{(i)}$ that were not fixed before. Thus, the hash function values $h_1(x), h_2(x)$, $x \in T - T^*$, are distributed fully randomly as well and are independent of those with $x \in T^*$. \square

3 Graph Properties and Basic Setup

For $m \in \mathbb{N}$ let \mathcal{G}_m denote the set of all bipartite (multi-)graphs with vertex set $[m]$ on each side of the bipartition. A set $\mathcal{A} \subseteq \mathcal{G}_m$ is called a *graph property*. For example, \mathcal{A} could be the set of graphs in \mathcal{G}_m that have excess larger than s . For a graph property $\mathcal{A} \subseteq \mathcal{G}_m$ and $T \subseteq U$, let \mathcal{A}_T denote the event that $G(T, h_1, h_2)$ has property \mathcal{A} (i.e., that $G(T, h_1, h_2) \in \mathcal{A}$). In the following, our main objective is to bound the probability $\Pr(\exists T \subseteq S: \mathcal{A}_T)$ for graph properties \mathcal{A} which are important for our analysis.

For the next lemma we need the following definitions. For $S \subseteq U$ and a graph property \mathcal{A} let $B_S^{\mathcal{A}} \subseteq \mathcal{Z}$ be the event $\exists T \subseteq S: \mathcal{A}_T \cap \text{bad}_T$ (see Def. 2). Considering fully random hash functions (h_1^*, h_2^*) for a moment, let $p_T^{\mathcal{A}} = \Pr(G(T, h_1^*, h_2^*) \in \mathcal{A})$.

Lemma 3. *For an arbitrary graph property \mathcal{A} we have*

$$\Pr(\exists T \subseteq S: \mathcal{A}_T) \leq \Pr(B_S^{\mathcal{A}}) + \sum_{T \subseteq S} p_T^{\mathcal{A}}. \quad (2)$$

Proof. $\Pr(\exists T \subseteq S: \mathcal{A}_T) \leq \Pr(B_S^{\mathcal{A}}) + \Pr((\exists T \subseteq S: \mathcal{A}_T) \cap \overline{B_S^{\mathcal{A}}})$, and

$$\sum_{T \subseteq S} \Pr(\mathcal{A}_T \cap \overline{B_S^{\mathcal{A}}}) \stackrel{(i)}{\leq} \sum_{T \subseteq S} \Pr(\mathcal{A}_T \cap \text{good}_T) \leq \sum_{T \subseteq S} \Pr(\mathcal{A}_T \mid \text{good}_T) \stackrel{(ii)}{=} \sum_{T \subseteq S} p_T^{\mathcal{A}},$$

where (i) holds by the definition of $B_S^{\mathcal{A}}$, and (ii) holds by Lemma 2(b). \square

This lemma encapsulates our overall strategy for bounding $\Pr(\exists T \subseteq S: \mathcal{A}_T)$. The second summand in (2) can be bounded assuming full randomness. The task of bounding the first summand is tackled separately, in Section 4.

4 A Bound for Leafless Graphs

The following observation, which is immediate from the definitions, will be helpful in applying Lemma 3.

Lemma 4. *Let $m \in \mathbb{N}$, and let $\mathcal{A} \subseteq \mathcal{A}' \subseteq \mathcal{G}_m$. Then $\Pr(B_S^{\mathcal{A}}) \leq \Pr(B_S^{\mathcal{A}'}).$ \square*

We define a graph property to be used in the role of \mathcal{A}' in applications of Lemma 4. A node with degree 1 in a graph is called a *leaf*; an edge incident with a leaf is called a *leaf edge*. An edge is called a *cycle edge* if removing it does not disconnect any two nodes. A graph is called *leafless* if it has no leaves. Let $\text{LL} \subseteq \mathcal{G}_m$ be the set of all leafless graphs. The *2-core* of a graph is its (unique) maximum leafless subgraph. The purpose of the present section is to prove a bound on $\Pr(B_S^{\text{LL}})$.

Lemma 5. *Let $\varepsilon > 0$, let $S \subseteq U$ with $|S| = n$, and let $m = (1 + \varepsilon)n$. Assume (h_1, h_2) is chosen at random from $\mathcal{Z} = \mathcal{Z}_{\ell, m}^{2k, c}$. Then $\Pr(B_S^{\text{LL}}) = O(n/\ell^{ck})$.*

We recall a standard notion from graph theory (already used in [9]; cf. App. A.1): The *cyclomatic number* $\gamma(G)$ of a graph G is the smallest number of edges one has to remove from G to obtain a graph with no cycles. Also, let $\zeta(G)$ denote the number of connected components of G (ignoring isolated points).

Lemma 6. *Let $N(t, \ell, \gamma, \zeta)$ be the number of non-isomorphic (multi-)graphs with ζ connected components and cyclomatic number γ that have t edges, ℓ of which are leaf edges. Then $N(t, \ell, \gamma, \zeta) = t^{O(\ell + \gamma + \zeta)}$.*

Proof. In [9, Lemma 2] it is shown that $N(t, \ell, \gamma, 1) = t^{O(\ell + \gamma)}$. Now note that each graph G with cyclomatic number γ , ζ connected components, $t - \ell$ non-leaf edges, and ℓ leaf edges can be obtained from some connected graph G' with cyclomatic number γ , $t - \ell + \zeta - 1$ non-leaf edges, and ℓ leaf edges by removing $\zeta - 1$ non-leaf, non-cycle edges. There are no more than $(t - \ell + \zeta - 1)^{\zeta - 1}$ ways for choosing the edges to be removed. This implies, using [9, Lemma 2]:

$$\begin{aligned} N(t, \ell, \gamma, \zeta) &\leq N(t + \zeta - 1, \ell, \gamma, 1) \cdot (t - \ell + \zeta - 1)^{\zeta - 1} \\ &\leq (t + \zeta)^{O(\ell + \gamma)} \cdot (t + \zeta)^{\zeta} = (t + \zeta)^{O(\ell + \gamma + \zeta)} = t^{O(\ell + \gamma + \zeta)}. \end{aligned} \quad \square$$

We shall need more auxiliary graph properties: A graph from \mathcal{G}_m belongs to LCY if at most one connected component contains leaves (the *leaf component*); for $K \geq 1$ it belongs to $\text{LCY}^{(K)}$ if it has the following four properties:

1. at most one connected component of G contains leaves (i. e., $\text{LCY}^{(K)} \subseteq \text{LCY}$);
2. the number $\zeta(G)$ of connected components is bounded by K ;
3. if present, the leaf component of G contains at most K leaf and cycle edges;
4. the cyclomatic number $\gamma(G)$ is bounded by K .

Lemma 7. *If $T \subseteq U$ and (h_1, h_2) is from \mathcal{Z} such that $G(T, h_1, h_2) \in \mathbf{LL}$ and (h_1, h_2) is T -bad, then there exists a subset T' of T such that $G(T', h_1, h_2) \in \mathbf{LCY}^{(4ck)}$ and (h_1, h_2) is T' -critical.*

Proof. Fix T and (h_1, h_2) as in the assumption. Initialize T' as T . We will remove (“peel”) edges from $G(T', h_1, h_2)$ in four stages. Of course, by “removing edge $(h_1(x), h_2(x))$ from $G(T', h_1, h_2)$ ” we mean removing x from T' .

Stage 1: Initially, we have $d_{T'}((h_1, h_2)) > k$. Repeat the following step: If $G(T', h_1, h_2)$ contains a leaf, remove a leaf edge from it, otherwise remove a cycle edge. Clearly, such steps maintain the property that $G(T', h_1, h_2)$ belongs to \mathbf{LCY} . Since $d_{T'}((h_1, h_2))$ can decrease by at most 1 when an edge is removed, we finally reach a situation where $d_{T'}((h_1, h_2)) = k$, i. e., (h_1, h_2) is T' -critical. Then $G(T', h_1, h_2)$ satisfies Property 1 from the definition of $\mathbf{LCY}^{(4ck)}$.

To prepare for the next stages, we define a set $T^* \subseteq T'$ with $2k \leq |T^*| \leq 2ck$, capturing keys that have to be “protected” during the following stages to maintain criticality of T' . If $|T'| = 2k$, we simply let $T^* = T'$. Then (h_1, h_2) is T^* -critical. If $|T'| > 2k$, a little more work is needed. By the definition of $d_T((h_1, h_2))$ we have $|T'| - \max\{|g_1(T')|, \dots, |g_c(T')|\} = k$. For each $j \in \{1, \dots, c\}$ Lemma 2(a) gives us a set $T_j^* \subseteq T'$ such that $|T_j^*| = 2k$ and $|g_j(T_j^*)| \leq k$. Let $T^* := T_1^* \cup \dots \cup T_c^*$. Clearly, $2k \leq |T^*| \leq 2ck$. Since $T_j^* \subseteq T^*$, we have $|T^*| - |g_j(T^*)| \geq |T_j^*| - |g_j(T_j^*)| \geq k$, for $1 \leq j \leq c$, and hence $d_{T^*}((h_1, h_2)) \geq k$. On the other hand we know that there exists some j with $|T'| - |g_j(T')| = k$. Since $T^* \subseteq T'$, we have $|T^*| - |g_j(T^*)| \leq k$ for this j . Altogether we get $d_{T^*}((h_1, h_2)) = k$, which means that (h_1, h_2) is T^* -critical also in this case.

Now we “mark” all edges of $G = G(T', h_1, h_2)$ whose keys belong to T^* .

Stage 2: Remove all components of G without marked edges. Afterwards there are at most $2ck$ components left, and G satisfies Property 2.

Stage 3: If G has a leaf component C , repeatedly remove unmarked leaf and cycle edges from C , while C has such edges. The remaining leaf and cycle edges in C are marked, and thus their number is at most $2ck$; Property 3 is satisfied.

Stage 4: If there is a leaf component C with z marked edges (where $z \leq 2ck$), then $\gamma(C) \leq z - 1$. Now consider a leafless component C' with cyclomatic number z . We need the following claim, which is proved in Appendix B.

Claim. Every leafless connected graph with i marked edges has a leafless connected subgraph with cyclomatic number $\leq i + 1$ that contains all marked edges.

This claim gives us a leafless subgraph C'' of C' with $\gamma(C'') \leq z + 1$ that contains all marked edges of C' . We remove from G all vertices and edges of C' that are not in C'' . Doing this for all leafless components yields the final key set T' and the final graph $G = G(T', h_1, h_2)$. Summing contributions to the cyclomatic number of G over all (at most $2ck$) connected components, we see that $\gamma(G) \leq 4ck$; Property 4 is satisfied. \square

What have we achieved? By Lemma 7, we can bound $\Pr(B_S^{\mathbf{LL}})$ by just adding, over all $T' \subseteq S$, the probabilities $\Pr(\mathbf{LCY}_{T'}^{(4ck)} \cap \text{crit}_{T'})$, that means, the terms $\Pr(\mathbf{LCY}_{T'}^{(4ck)} \mid \text{crit}_{T'}) \cdot \Pr(\text{crit}_{T'})$. Lemma 2(a) takes care of the second factor. By

Lemma 2(b), we may assume that (h_1, h_2) acts fully random on T' for the first factor. The next lemma estimates this factor, using the notation from Section 3.

Lemma 8. *Let $T \subseteq U$, $|T| = t$, and $c, k \geq 1$. Then $p_T^{\text{LCY}^{(4ck)}} \leq t! \cdot t^{O(1)} / m^{t-1}$.*

Proof. By Lemma 6, there are at most $t^{O(ck)} = t^{O(1)}$ ways to choose a bipartite graph G in $\text{LCY}^{(4ck)}$ with t edges. Graph G cannot have more than $t + 1$ nodes, since cyclic components have at most as many nodes as edges, and in the single leaf component, if present, the number of nodes is at most one bigger than the number of edges. In each component of G , there are two ways to assign the vertices to the two sides of the bipartition. After such an assignment is fixed, there are at most m^{t+1} ways to label the vertices with elements of $[m]$, and there are $t!$ ways to label the edges of G with the keys in T . Assume now such labels have been chosen for G . Draw t edges $(h_1^*(x), h_2^*(x))$ from $[m]^2$ uniformly at random. The probability that they exactly fit the labeling of nodes and edges of G is $1/m^{2t}$. Thus, $p_T^{\text{LCY}^{(4ck)}} \leq m^{t+1} \cdot t! \cdot t^{O(1)} / m^{2t} = t! \cdot t^{O(1)} / m^{t-1}$. \square

We can now finally prove Lemma 5, the main lemma of this section.

Proof (of Lemma 5). By Lemma 7, and using the union bound, we get

$$\begin{aligned} \Pr(B_S^{\text{LL}}) &= \Pr(\exists T \subseteq S : \text{LL}_T \cap \text{bad}_T) \leq \Pr(\exists T' \subseteq S : \text{LCY}_{T'}^{(4ck)} \cap \text{crit}_{T'}) \\ &\leq \sum_{T' \subseteq S} \Pr(\text{LCY}_{T'}^{(4ck)} \mid \text{crit}_{T'}) \cdot \Pr(\text{crit}_{T'}) =: \rho_S. \end{aligned}$$

By Lemma 2(b), given the event that (h_1, h_2) is T' -critical, (h_1, h_2) acts fully random on T' . Using Lemma 8 and Lemma 2(a), this yields:

$$\Pr(\text{LCY}_{T'}^{(4ck)} \mid \text{crit}_{T'}) \cdot \Pr(\text{crit}_{T'}) \leq (|T'|! \cdot |T'|^{O(1)} / m^{|T'|-1}) \cdot (|T'|^2 / \ell)^{ck}.$$

Summing up, collecting sets T' of equal size together, and using that ck is constant, we obtain

$$\rho_S \leq \sum_{2k \leq t \leq n} \binom{n}{t} \cdot \frac{t! \cdot t^{O(1)}}{m^{t-1}} \cdot \left(\frac{t^2}{\ell}\right)^{ck} \leq \frac{n}{\ell^{ck}} \cdot \sum_{2k \leq t \leq n} \frac{t^{O(1)}}{(1+\varepsilon)^{t-1}} = O\left(\frac{n}{\ell^{ck}}\right). \quad \square$$

5 Cuckoo Hashing With a Stash

In this section we prove the desired bound on the rehash probability of cuckoo hashing with a stash when functions from \mathcal{Z} are used. We focus on the question whether the pair (h_1, h_2) allows storing key set S in the two tables with a stash of size s . In view of Lemma 1, we identify minimal graphs with excess $s + 1$.

Definition 3. *An excess- $(s + 1)$ core graph is a leafless graph G with excess exactly $s + 1$ in which all connected components have at least two cycles. By $\text{CS}^{(s+1)}$ we denote the set of all excess- $(s + 1)$ core graphs in \mathcal{G}_m .*

Lemma 9. *Let $G = G(S, h_1, h_2)$ be a cuckoo graph with $\text{ex}(G) \geq s + 1$. Then G contains an excess- $(s + 1)$ core graph as a subgraph.*

Proof. We repeatedly remove edges from G . First we remove cycle edges until the excess is exactly $s + 1$. Then we remove components that are trees or unicyclic. Finally we remove leaf edges one by one until the remaining graph is leafless. \square

We are now ready to state our main theorem.

Theorem 1. *Let $\varepsilon > 0$ and $0 < \delta < 1$, let $s \geq 0$ and $k \geq 1$ be given. Assume $c \geq (s + 2)/(\delta k)$. For $n \geq 1$ consider $m \geq (1 + \varepsilon)n$ and $\ell = n^\delta$. Let $S \subseteq U$ with $|S| = n$. Then for (h_1, h_2) chosen at random from $\mathcal{Z} = \mathcal{Z}_{\ell, m}^{2k, c}$ the following holds:*

$$\Pr(\text{ex}(G(S, h_1, h_2)) \geq s + 1) = O(1/n^{s+1}).$$

Proof. By Lemma 3 and Lemma 9, the probability that the excess of $G(S, h_1, h_2)$ is at least $s + 1$ is

$$\Pr(\exists T \subseteq S: \text{CS}_T^{(s+1)}) \leq \Pr(B_S^{\text{CS}^{(s+1)}}) + \sum_{T \subseteq S} p_T^{\text{CS}^{(s+1)}}. \quad (3)$$

Since $\text{CS}^{(s+1)} \subseteq \text{LL}$, we can combine Lemmas 4 and 5 to obtain $\Pr(B_S^{\text{CS}^{(s+1)}}) \leq \Pr(B_S^{\text{LL}}) = O(n/\ell^{ck})$. It remains to bound the second summand in (3).⁴

Lemma 10. $\sum_{T \subseteq S} p_T^{\text{CS}^{(s+1)}} = O(1/n^{s+1})$.

Proof. We start by counting (unlabeled) excess- $(s + 1)$ core graphs with t edges. A connected component C of such a graph G with cyclomatic number $\gamma(C)$ (which is at least 2) contributes $\gamma(C) - 1$ to the excess of G . This means that if G has $\zeta = \zeta(G)$ components, then $s + 1 = \gamma(G) - \zeta$ and $\zeta \leq s + 1$, and hence $\gamma = \gamma(G) \leq 2(s + 1)$. Using Lemma 6, there are at most $N(t, 0, \gamma, \zeta) = t^{O(\gamma + \zeta)} = t^{O(s)}$ such graphs G . If from each component C of such a graph G we remove $\gamma(C) - 1$ cycle edges, we get unicyclic components, which have as many nodes as edges. This implies that G has $t - (s + 1)$ nodes.

Now fix a bipartite (unlabeled) excess- $(s + 1)$ core graph G with t edges and ζ components, and let $T \subseteq U$ with $|T| = t$ be given. There are $2^\zeta \leq 2^{s+1}$ ways of assigning the $t - s - 1$ nodes to the two sides of the bipartition, and then at most m^{t-s-1} ways of assigning labels from $[m]$ to the nodes. Thus, the number of bipartite graphs with property $\text{CS}^{(s+1)}$, where each node is labeled with one side of the bipartition and an element of $[m]$, and where the t edges are labeled with distinct elements of T is smaller than $t! \cdot 2^{s+1} \cdot m^{t-s-1} \cdot t^{O(s)}$.

Now if G with such a labeling is fixed, and we choose t edges from $[m]^2$ uniformly at random, the probability that all edges $(h_1(x), h_2(x))$, $x \in T$, match

⁴ We remark that the following calculations also give an alternative, simpler proof of [11, Theorem 2.1] for the fully random case, even if the effort needed to prove Lemma 6 and [9, Lemma 2] is taken into account.

the labeling is $1/m^{2t}$. For constant s , this yields the following bound:

$$\begin{aligned} \sum_{T \subseteq S} p_T^{\text{CS}^{(s+1)}} &\leq \sum_{s+3 \leq t \leq n} \binom{n}{t} \frac{2^{s+1} \cdot n^{t-s-1} \cdot t! \cdot t^{O(s)}}{m^{2t}} \leq \frac{2^{s+1}}{n^{s+1}} \cdot \sum_{s+3 \leq t \leq n} \frac{n^t \cdot t^{O(1)}}{m^t} \\ &= O\left(\frac{1}{n^{s+1}}\right) \cdot \sum_{s+3 \leq t \leq n} \frac{t^{O(1)}}{(1+\varepsilon)^t} = O\left(\frac{1}{n^{s+1}}\right). \quad \square \end{aligned}$$

Since $\ell = n^\delta$ and $c \geq (s+2)/(k\delta)$, we get

$$\Pr(\text{ex}(G) \geq s+1) = O(n/\ell^{ck}) + O(1/n^{s+1}) = O(1/n^{s+1}). \quad \square$$

6 Simulating Uniform Hashing

In the following, let R be the range of the hash function to construct, and assume that (R, \oplus) is a commutative group. (We could use $R = [t]$ with addition mod t .)

Theorem 2. *Let $n \geq 1$, $0 < \delta < 1$, $\varepsilon > 0$, and $s \geq 0$ be given. There exists a data structure DS_n that allows us to compute a function $h: U \rightarrow R$ such that:*

- (i) *For each $S \subseteq U$ of size n there is an event B_S of probability $O(1/n^{s+1})$ such that conditioned on $\overline{B_S}$ the function h is distributed uniformly on S .*
- (ii) *For arbitrary $x \in U$, $h(x)$ can be evaluated in time $O(s/\delta)$.*
- (iii) *DS_n comprises $2(1+\varepsilon)n + O(sn^\delta)$ words from R and $O(s)$ words from U .*

Proof. Let $k \geq 1$ and choose $c \geq (s+2)/(k\delta)$. Given U and n , set up DS_n as follows. Let $m = (1+\varepsilon)n$ and $\ell = n^\delta$, and choose and store a hash function pair (h_1, h_2) from $\mathcal{Z} = \mathcal{Z}_{\ell, m}^{2k, c}$ (see Definition 1), with component functions g_1, \dots, g_c from \mathcal{H}_ℓ^{2k} . In addition, choose 2 random vectors $t_1, t_2 \in R^m$, c random vectors $y_1, \dots, y_c \in R^\ell$, and choose f at random from a $2k$ -wise independent family of hash functions from U to R .

Using DS_n , the mapping $h: U \rightarrow R$ is defined as follows:

$$h(x) = t_1[h_1(x)] \oplus t_2[h_2(x)] \oplus f(x) \oplus y_1[g_1(x)] \oplus \dots \oplus y_c[g_c(x)].$$

DS_n satisfies (ii) and (iii) of Theorem 2. We show that it satisfies (i) as well.

First, consider only the hash functions (h_1, h_2) from \mathcal{Z} . By Lemma 5 we have $\Pr(B_S^{\text{LL}}) = O(n/\ell^{ck}) = O(1/n^{s+1})$. Now fix $(h_1, h_2) \notin B_S^{\text{LL}}$, which includes fixing the components g_1, \dots, g_c . Let $T \subseteq S$ be such that $G(T, h_1, h_2)$ is the 2-core of $G(S, h_1, h_2)$, the maximal subgraph with minimum degree 2. Graph $G(T, h_1, h_2)$ is leafless, and since $(h_1, h_2) \notin B_S^{\text{LL}}$, we have that (h_1, h_2) is T -good. Now we note that the part $f(x) \oplus \bigoplus_{1 \leq j \leq c} y_j[g_j(x)]$ of $h(x)$ acts exactly as one of our hash functions h_1 and h_2 (see Definition 1(a)), so that arguing as in the proof of Lemma 2 we see that $h(x)$ is fully random on T .

Now assume that f and the entries in the tables y_1, \dots, y_c are fixed. It is not hard to show that the random entries in t_1 and t_2 alone make sure that $h(x)$, $x \in S - T$, is fully random. (Such proofs were given in [9] and [16].) \square

Concluding Remarks

We presented a family of efficient hash functions and showed that it exhibits sufficiently strong random properties to run cuckoo hashing with a stash, preserving the favorable performance guarantees of this hashing scheme. We also described a simple construction for simulating uniform hashing. We remark that the performance of our construction can be improved by using 2-universal hash families⁵ (see, e.g., [2,5]) for the g_j -components. The proof of Lemma 2 can be adapted easily to these weaker families. It remains open whether generalized cuckoo hashing [10,8] can be run with efficient hash families.

References

1. Calkin, N.J.: Dependent sets of constant weight binary vectors. *Combinatorics, Probability and Computing* **6**(3), 263–271 (1997)
2. Carter, L., Wegman, M.N.: Universal classes of hash functions. *J. Comput. Syst. Sci.* **18**(2), 143–154 (1979)
3. Devroye, L., Morin, P.: Cuckoo hashing: Further analysis. *Inf. Process. Lett.* **86**(4), 215–219 (2003)
4. Diestel, R.: *Graph Theory*. Springer (2005)
5. Dietzfelbinger, M., Hagerup, T., Katajainen, J., Penttonen, M.: A reliable randomized algorithm for the closest-pair problem. *J. Algorithms* **25**(1), 19–51 (1997)
6. Dietzfelbinger, M., Rink, M.: Applications of a splitting trick. In: Proc. 36th ICALP (1). pp. 354–365. LNCS 5555, Springer (2009)
7. Dietzfelbinger, M., Schellbach, U.: On risks of using cuckoo hashing with simple universal hash classes. In: Proc. 20th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA). pp. 795–804 (2009)
8. Dietzfelbinger, M., Weidling, C.: Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.* **380**(1-2), 47–68 (2007)
9. Dietzfelbinger, M., Woelfel, P.: Almost random graphs with simple hash functions. In: Proc. 35th ACM Symp. on Theory of Computing (STOC). pp. 629–638. New York, NY, USA (2003)
10. Fotakis, D., Pagh, R., Sanders, P., Spirakis, P.G.: Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.* **38**(2), 229–248 (2005)
11. Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: Cuckoo hashing with a stash. In: Proc. 16th ESA 2008. pp. 611–622. LNCS 5193, Springer (2008)
12. Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.* **39**(4), 1543–1561 (2009)
13. Klassen, T.Q., Woelfel, P.: Independence of tabulation-based hash classes. In: Proc. 10th LATIN 2012. pp. 506–517. LNCS 7256, Springer (2012)
14. Kutzelnigg, R.: A further analysis of cuckoo hashing with a stash and random graphs of excess r . *Discr. Math. and Theoret. Comput. Sci.* **12**(3), 81–102 (2010)
15. Mitzenmacher, M., Vadhan, S.P.: Why simple hash functions work: exploiting the entropy in a data stream. In: Proc. 19th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA). pp. 746–755 (2008)

⁵ A family \mathcal{H} of hash functions with range R is 2-universal if for each pair $x, y \in U$, $x \neq y$, and h chosen at random from \mathcal{H} we have $\Pr(h(x) = h(y)) \leq 2/|R|$.

16. Pagh, A., Pagh, R.: Uniform hashing in constant time and optimal space. *SIAM J. Comput.* **38**(1), 85–96 (2008)
17. Pagh, R., Rodler, F.F.: Cuckoo hashing. *J. Algorithms* **51**(2), 122–144 (2004)
18. Pătraşcu, M., Thorup, M.: The power of simple tabulation hashing. In: Proc. 43rd ACM Symp. on Theory of Computing (STOC). pp. 1–10 (2011)
19. Siegel, A.: On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.* **33**(3), 505–543 (2004)
20. Thorup, M., Zhang, Y.: Tabulation based 4-universal hashing with applications to second moment estimation. In: Proc. 15th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA). pp. 615–624 (2004)
21. Thorup, M., Zhang, Y.: Tabulation based 5-universal hashing and linear probing. In: Proc. 12th ALENEX. pp. 62–76. SIAM (2010)
22. Wegman, M.N., Carter, L.: New classes and applications of hash functions. In: Proc. 20th Ann. Symp. on Foundations of Computer Science (FOCS). pp. 175–182. IEEE Computer Society (1979)
23. Woelfel, P.: Asymmetric balanced allocation with simple hash functions. In: Proc. 17th ACM-SIAM Symp. on Discrete Algorithms (SODA). pp. 424–433 (2006)

A Excess, Stash Size, and Insertions

In this supplementary section, provided for the convenience of the reader, we clarify the connection between stash size needed and the excess $\text{ex}(G(S, h_1, h_2))$ of the cuckoo graph $G(S, h_1, h_2)$ as well as the role of insertion procedures. In particular, we prove Lemma 1. The central statements of this section can also be found in [12,14].

A.1 The Excess of a Graph

For G a graph, $\zeta(G)$ denotes the number of connected components of G . The cyclomatic number $\gamma(G)$, technically defined as “the dimension of the cycle space of G ”, can be characterized by the following basic formula [4]:

$$\gamma(G) = m - n + \zeta(G), \quad (4)$$

for n the number of nodes and m the number of edges of G . Note that acyclic graphs are characterized by the equation $n = m + \zeta(G)$ and hence by the equation $\gamma(G) = 0$. Using (4), two helpful ways of viewing $\gamma(G)$ are easy to prove.

- Lemma 11.** (a) *If we remove edges from G sequentially, in an arbitrary order, and the resulting graph is acyclic, then $\gamma(G)$ is the number of cycle edges removed.*
- (b) *$\gamma(G)$ is the minimum number of edges one has to remove from G such that the resulting graph is acyclic.*

Proof. Assume a subgraph G' of G (with all n nodes) has $m' > 0$ edges. If we remove one edge e' from G' to obtain G'' , we have, using (4) twice:

$$\gamma(G'') = (m' - 1) - n + \zeta(G'') = \gamma(G') - (1 - (\zeta(G'') - \zeta(G'))).$$

We observe:

- If e' is a cycle edge in G' , then $\zeta(G'') = \zeta(G')$, and hence $\gamma(G'') = \gamma(G') - 1$.
- If e' is not a cycle edge, then $\zeta(G'') = \zeta(G') + 1$, and hence $\gamma(G'') = \gamma(G')$.

We prove (a): By what we just observed, to reduce the cyclomatic number from $\gamma(G)$ to 0 the number of rounds in which a cycle edge is removed must be $\gamma(G)$. Now we prove (b): Think of the edges as being removed sequentially. Again, by our observation, in order to reduce the cyclomatic number from $\gamma(G)$ to 0 by removing as few edges as possible we should never remove an edge that is not on a cycle. In this way we remove exactly $\gamma(G)$ (cycle) edges. \square

We have defined the excess $\text{ex}(G)$ of a graph G as the minimum number of edges one has to remove from G so that the remaining subgraph has only acyclic and unicyclic components. In [14] the characterization of this quantity given next was used as a definition; the same idea was used in [12] (without giving it a name).

For G a graph, let $\zeta_{\text{cyc}}(G)$ denote the number of cyclic components of G .

Lemma 12. *In all graphs G the equation $\text{ex}(G) = \gamma(G) - \zeta_{\text{cyc}}(G)$ is satisfied.*

Proof. Assume G has n nodes and m edges.

“ \leq ”: Starting with G , we iteratively remove *cycle* edges until each cyclic component has only one cycle left. The number of edges removed is at least $\text{ex}(G)$. Call the resulting graph G' . Removing one cycle edge from each of the $\zeta_{\text{cyc}}(G)$ cyclic components of G' will yield an acyclic graph. Lemma 11(a) tells us that together exactly $\gamma(G)$ edges have been removed; hence $\gamma(G) \geq \text{ex}(G) + \zeta_{\text{cyc}}(G)$. “ \geq ”: Choose a set E^+ of $\text{ex}(G)$ edges in G such that removing these edges leaves a graph G' with only acyclic and unicyclic components. Now imagine that the edges in E^+ are removed one by one in an arbitrary order. Let β denote the number of edges in E^+ that are on a cycle when removed; the other $\text{ex}(G) - \beta$ many were non-cycle edges when removed. Removing one cycle edge from each cyclic component of G' will leave an acyclic graph. Counting the number of cycle edges we removed altogether, and applying Lemma 11(a) again, we see that $\gamma(G) = \beta + \zeta_{\text{cyc}}(G')$. Since removing a non-cycle edge from a graph can increase the number of cyclic components by at most 1, we have that $\zeta_{\text{cyc}}(G') \leq \zeta_{\text{cyc}}(G) + (\text{ex}(G) - \beta)$. Combining the inequalities yields $\gamma(G) \leq \zeta_{\text{cyc}}(G) + \text{ex}(G)$. \square

A.2 The Excess of the Cuckoo Graph and the Stash Size

The purpose of this section is to prove Lemma 1, which we recall here. We assume that h_1 and h_2 are given, and write $G(S)$ for $G(S, h_1, h_2)$, for $S \subseteq U$.

Lemma 1 ([12]). *The keys from S can be stored in the two tables and a stash of size s using (h_1, h_2) if and only if $\text{ex}(G(S)) \leq s$.*

Proof. “ \Rightarrow ”: Assume T is a subset of S of size at most s such that all keys from $S' = S - T$ can be stored in the two tables. Then all components of $G(S')$ must be acyclic or unicyclic. (Assume C is a component with $\gamma(C) > 1$. Then by (4)

the number of edges (keys) in C would be strictly larger than the number of nodes (table positions), which is impossible.) Since $G(S')$ is obtained from $G(S)$ by removing the edges $(h_1(x), h_2(x))$, $x \in T$, we get $\text{ex}(G(S)) \leq s$.

“ \Leftarrow ”: Assume $\text{ex}(G(S)) \leq s$. Choose a subset T of S of size $\text{ex}(G(S))$ such that $G(S - T)$ has only acyclic and unicyclic components. From what is known about the behaviour of standard cuckoo hashing, we can store $S' = S - T$ in the two tables using h_1 and h_2 (e.g., see [3, Sect. 4]). (This can even be proved directly. If one of the nodes touched by an edge $(h_1(x), h_2(x))$, $x \in S'$, has degree 1, we place x in the corresponding cell. Iterating this, we can place all keys excepting those that belong to cycle edges. Since $G(S')$ has only acyclic and unicyclic components, the cycle edges form isolated simple cycles, and clearly the keys that belong to such a cycle can be placed in the corresponding cells.) By assumption, the keys from T fit into the stash. \square

A.3 The Insertion Procedure

We consider here the obvious generalization of the insertion procedure in standard cuckoo hashing [17]. It assumes that a procedure *rehash* is given that will choose two new hash functions and insert all keys anew. The parameter *maxloop* is used for avoiding infinite loops. (When using cuckoo hashing with a stash of size s , one will choose $\text{maxloop} = \Theta((s + 2) \log n)$. For analysis purposes, larger values of *maxloop* are considered as well.) Empty table cells contain **nil**. The operation *swap* exchanges the contents of two variables.

Algorithm 1 (Insertion in a cuckoo table with a stash).

```

procedure stashInsert( $x$ : key)
(1)   nestless :=  $x$ ;
(2)    $i$  := 1;
(3)   repeat maxloop times
(4)     swap(nestless,  $T_i[h_i(\text{nestless})]$ );
(5)     if nestless = nil then return;
(6)      $i$  :=  $3 - i$ ;
(7)   if stash is not yet full
(8)     then add nestless to stash
(9)     else rehash.
```

As long as it is not finished, the procedure maintains a “nestless” key (in **nestless**) and the current index $i \in \{1, 2\}$ (in i) of the table where this key is to be placed. When a new key x is to be inserted, it is declared “nestless” and i is set to 1. As long as there is a nestless key x , but at most for *maxloop* rounds, the following is iterated: Assume x is nestless and the current index is i . Then x is placed in position $h_i(x)$ in table T_i . If this position is empty, the procedure terminates; if it contains a key x' , that key gets evicted to make room for x , is declared nestless, and i is changed to the other value $3 - i$. If the loop does not terminate within *maxloop* rounds, the key that is currently nestless gets stored in the stash. If this causes the stash to overflow, a *rehash* is carried out. (This

may be realized by collecting all keys from tables and stash as well as the nestless key, choosing a new pair (h_1, h_2) of hash functions, and calling the insertion procedure for all keys.)

A.4 Complete Insertion Loops and the Excess

We first look at the behavior of certain variants of the insertion procedure (called “complete”), which exhibit the following behavior when x is inserted: (i) if with *maxloop* set to infinity the loop were to run forever, then this is noticed and at some point the currently nestless key is put in the stash; (ii) otherwise the loop is left to run until the nestless key is stored in an empty cell. It is not hard to see (cf. [3]) that one obtains a complete variant from Algorithm 1 if one chooses *maxloop* as some number larger than $2|S| + 3$.

Proposition 1 ([12,14]). *If inserting the keys of S by some complete insertion procedure places s keys in the stash, then $s = \text{ex}(G(S)) = \text{ex}(G(S, h_1, h_2))$.*

Proof. “ \geq ”: After the insertion is complete, all keys from S are stored in the two tables and the stash. Lemma 1 implies that $s \geq \text{ex}(G(S))$.

“ \leq ”: For this, we use induction on the size of S . If $S = \emptyset$, excess and stash size are both 0. Now assume as induction hypothesis that set S has been inserted, that the set of keys placed in the stash is T , and that $|T| = s \leq \text{ex}(G(S))$. Let $S' = S - T$. We insert a new key y from $U - S$.

Case 1: The insertion procedure finds that y can be accommodated without using the stash.—The stash size remains s , and $s \leq \text{ex}(G(S)) \leq \text{ex}(G(S \cup \{y\}))$.

Case 2: The complete insertion procedure notices that the loop were to run forever and places some key in the stash.—By the properties of the complete insertion loop for standard cuckoo hashing as explored in [3] we know that $G(S' \cup \{y\})$ must contain a connected component that is neither acyclic nor unicyclic. Since $\text{ex}(G(S')) = 0$, it must be edge $(h_1(x), h_2(x))$ that makes the difference. This means that each endpoint of $(h_1(x), h_2(x))$ lies in some cyclic component of $\text{ex}(G(S'))$. Now $G(S')$ is a subgraph of $G(S)$, so the same is true in $G(S)$. Recall Lemma 12, and consider two cases when changing from S to $S \cup \{y\}$: If the endpoints of $(h_1(x), h_2(x))$ lie in two different cyclic components of $G(S)$, then the number of cyclic components decreases by 1, hence the excess increases by 1; if they lie in one and the same cyclic component, then the cyclomatic number increases by 1, and the excess increases by 1 as well. In both cases we get that $s + 1 \leq \text{ex}(G(S)) + 1 = \text{ex}(G(S \cup \{y\}))$. \square

A.5 Standard Insertion

It turns out that by choosing $\text{maxloop} = \Theta((s + 2) \log n)$ in Algorithm 1 we can make sure that with probability of $O(1/n^{s+1})$ no rehash is necessary.⁶ Note that

⁶ If deletions are allowed, before calling *rehash* one should try whether any one of the s keys presently stored in the stash can be inserted into the tables by the insertion procedure. We ignore deletions here.

if the stash has size 0, then Algorithm 1 is exactly the insertion procedure of standard cuckoo hashing from [17]. The following claim and proof are similar to what has to be done in the analysis of standard cuckoo hashing.

Proposition 2. *Assume the hash functions (h_1^*, h_2^*) are fully random, and the keys from S are inserted sequentially into a cuckoo table with a stash of size s , using Algorithm 1. If we choose $\text{maxloop} = \alpha(s+2) \log n$ for a suitable constant $\alpha > 0$, then we have:*

$$\Pr(\text{the stash of size } s \text{ overflows}) = O(1/n^{s+1}).$$

Sketch of proof. Theorem 1 tells us that the probability that a stash of size s is not sufficient because the excess of $G(S, h_1^*, h_2^*)$ is too large is $O(1/n^{s+1})$. All we have to show is that the probability is also this small that an extra key slips into the stash because the insertion loop was stopped by the step counter hitting maxloop . Let $S = \{x_1, \dots, x_n\}$, with the keys listed in the order in which they are inserted, and let $S_j = \{x_1, \dots, x_j\}$. For $1 \leq j \leq n$ and some bound p consider the event that the insertion procedure for x_j needs p or more rounds. One can show (this was done in [17] with a different terminology) that then $G(S_j)$ must contain a path $u_0, u_1, \dots, u_t, u_{t+1}$, with $t = \lceil p/3 \rceil$, where u_0 is the node corresponding to $T_i[h_i(x_j)]$, for $i = 1$ or $i = 2$, and u_0, \dots, u_t are distinct nodes. Viewing the situation in terms of edges this means that there must be some $T \subseteq S_{j-1}$ of size $t = \lceil p/3 \rceil$ such that the edges $(h_1(x), h_2(x))$, with $x \in T$, in some order, form such a path. The latter event we call \mathcal{A}_T . We have $\Pr(\mathcal{A}_T) \leq t! \cdot 2/m^t$. The number of sets T to consider is $\binom{j-1}{t} < \binom{n}{t}$. Thus,

$$\Pr(\exists T \subseteq S_{j-1}: |T| = t \wedge \mathcal{A}_T) \leq \binom{n}{t} t! \cdot \frac{2}{m^t} < \frac{2}{(1+\varepsilon)^t}. \quad (5)$$

If the insertion of x_j increases the stash size although $\text{ex}(G(S_j)) = \text{ex}(G(S_{j-1}))$, then this insertion must make $p = \text{maxloop}$ steps. By (5), the probability of this to happen is smaller than $2/(1+\varepsilon)^{\lceil \text{maxloop}/3 \rceil}$. So, if we choose $\text{maxloop} = 3(s+2) \log_{1+\varepsilon} n$ (i. e., $\alpha = 3/\log(1+\varepsilon) = \Theta(1/\varepsilon)$), this probability will be smaller than $1/n^{s+2}$. Summing over all j we obtain the bound $O(1/n^{s+1})$. \square

Proposition 3. *Assume the hash functions (h_1^*, h_2^*) are fully random, the keys from S are stored in a cuckoo table with a stash of size s , and a new key y is inserted by Algorithm 1, with $\text{maxloop} \geq \alpha(s+2) \log n$ for α as in Proposition 2. Then the expected number of steps needed for this insertion is $O(1)$.*

Sketch of proof. Let the random variable Z denote the number of rounds needed for this insertion. We ignore the cost of a *rehash*. (The contribution of this rare event to the overall insertion time is $O(1/n^s)$. A discussion of the case $s = 0$ can be found in [9].) Then $E(Z) \leq \sum_{p \geq 1} \Pr(\text{at least } p \text{ rounds are needed to store } y)$, and hence, using (5) and arguing as in [17],

$$E(Z) \leq \sum_{p \geq 1} \Pr(\exists T: |T| = \lceil p/3 \rceil \wedge \mathcal{A}_T) \leq \sum_{p \geq 1} \frac{2}{(1+\varepsilon)^{\lceil p/3 \rceil}} = O(1). \quad \square$$

Of course, the last two propositions are formulated for fully random hash functions. Using the techniques developed for the proof of Theorem 1 one can show that they are valid for hash functions from $\mathcal{Z}_{\ell,m}^{2k,c}$ as well, for the parameter choices as in that theorem. The only difference is that instead of the graph property LL one has to use the graph property “connected” in a way explored in detail in [9].

B Proof of a Claim

We prove the following claim, stated and used in the proof of Lemma 7.

Claim. Every leafless connected graph with i marked edges has a leafless connected subgraph with cyclomatic number $\leq i+1$ that contains all marked edges.

Proof. Let $G = (V, E)$ be a leafless connected graph. If $\gamma(G) \leq i+1$, there is nothing to prove. Thus assume $\gamma(G) \geq i+2$. Choose an arbitrary spanning tree (V, E_0) of G .

There are two types of edges in G : *bridge edges* and *cycle edges*. A bridge edge is an edge whose deletion disconnects the graph, cycle edges are those whose deletion does not disconnect the graph.

Clearly, all bridge edges are in E_0 . Let $E_{\text{mb}} \subseteq E_0$ denote the set of marked bridge edges. Removing the edges of E_{mb} from G will split V into $|E_{\text{mb}}| + 1$ connected components $V_1, \dots, V_{|E_{\text{mb}}|+1}$; removing the edges of E_{mb} from the spanning tree (V, E_0) will give exactly the same components. For each *cyclic* component V_j we choose one edge $e_j \notin E_0$ that connects two nodes in V_j . The set of these $|E_{\text{mb}}| + 1$ edges is called E_1 . Now each marked bridge edge lies on a path connecting two cycles in $(V, E_0 \cup E_1)$.

Recall from graph theory [4] the notion of a fundamental cycle: Clearly, each edge $e \in E - E_0$ closes a unique cycle with E_0 . The cycles thus obtained are called the fundamental cycles of G w.r. t. the spanning tree (V, E_0) . Each cycle in G can be obtained as an XOR-combination of fundamental cycles. (This is just another formulation of the standard fact that the fundamental cycles form a basis of the “cycle space” of G , see [4].) From this it is immediate that every cycle edge of G lies on some fundamental cycle. Now we associate an edge $e' \notin E_0$ with each marked cycle edge $e \in E_{\text{mc}}$. Given e , let $e' \notin E_0$ be such that e is on the fundamental cycle of e' . Let E_2 be the set of all edges e' chosen in this way. Clearly, each $e \in E_{\text{mc}}$ is a cycle edge in $(V, E_0 \cup E_2)$.

Now let $G' = (V, E_0 \cup E_1 \cup E_2)$. Note that $|E_1 \cup E_2| \leq (|E_{\text{mb}}| + 1) + |E_{\text{mc}}| \leq i+1$ and thus $\gamma(G') \leq i+1$. In G' , each marked edge is on a cycle or on a path that connects two cycles. If we iteratively remove leaf edges from G' until no leaf is left, none of the marked edges will be affected. In this way we obtain the desired leafless subgraph G^* with $\gamma(G^*) = \gamma(G') \leq i+1$. \square