Discrete Mathematics and Theoretical Computer Science DMTCS vol. **12:3**, 2010, 81–102

A further analysis of Cuckoo Hashing with a Stash and Random Graphs of Excess r

Reinhard Kutzelnigg[†‡]

Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Wien, Austria

received 11th August 2009, *revised 21st March 2010*, *accepted 26th July 2010*.

Cuckoo hashing is a hash table data structure offering constant access time, even in the worst case. As a drawback, the construction fails with small, but practically significant probability. However, Kirsch et al. (2008) showed that a constant-sized additional memory, the so called stash, is sufficient to reduce the failure rate drastically. But so far, using a modified insertion procedure that demands additional running time to look for an admissible key is required. As a major contribution of this paper, we show that the same bounds on the failure probability hold even without this search process and thus, the performance increases. Second, we extend the analysis to simplified cuckoo hashing, a variant of the original algorithm offering increased performance. Further, we derive some explicit asymptotic approximations concerning the number of usual resp. bipartite graphs related to the data structures. Using these results, we obtain much more precise asymptotic expansions of the success rate. These calculations are based on a generating function approach and applying the saddle point method. Finally, we provide numerical results to support the theoretical analysis.

Keywords: Hashing, Cuckoo hashing, Algorithms, Generating functions, Random graphs, Saddle point method

1 Introduction

In computer science, hash tables are a frequently used tool to build dictionary-like data structures that support fast insertion, search and potentially also deletion operations, see, e.g. Cormen et al. (2001). All these algorithms are based on using hash functions that map data records (keys) to a unique memory cell of the table. We say that two different keys collide if both try to occupy a single memory slot. The critical point of each hash algorithm is the handling of colliding keys.

In particular, hash algorithms that offer constant access time even in the worst case are of high interest. One of these algorithms is cuckoo hashing that was first proposed by Pagh and Rodler (2004). In contrast to common hash techniques like open addressing and hashing with chaining (see, e.g., Knuth (1998)), collisions are resolved by rearranging keys. This is achieved by using two independent hash functions h_1 and

[†]Email: `reinhard.kutzelnigg@gmail.com`

[‡]The author was supported by the EU FP6-NEST-Adventure Programme, Contract number 028875 (NEMO), and by the Austrian Science Foundation FWF, project S9604.

h_2, both map a key to a unique position in the data structure. These are the only allowed storage locations of that key and, hence search operations require at most to access two memory cells. There are several different variants of cuckoo hashing known in the literature, see Fotakis et al. (2005); Dietzfelbinger and Weidling (2007) and Kutzelnigg (2010). In this paper we consider the following two: The standard algorithm introduced in Pagh and Rodler (2004) splits the available memory cells in two equal parts and grants each hash function exclusively access to one of these regions. However, this split-up is not necessary. We obtain an in some sense simplified algorithm, if both hash functions address the whole table. It is shown in Kutzelnigg (2009, 2010) that this variation, henceforth referred to as simplified cuckoo hashing, offers improved search and insertion performance.

Despite this difference, the insertion process works for both algorithms considered here as follows: To insert a new key x, we put it into the table position indicated by $h_1(x)$. Now, if this position was previously empty, the insertion is complete. Otherwise, we kick-out the key y that previously occupied $h_1(x) = h_1(y)$ and move it to its alternative position $h_2(y)$. If this position was already in use by a key z, we continue moving z to its alternative position. We carry on in this way until an empty cell is found or we detect an endless loop. The latter case can be uncovered by the number of kick-out steps performed. The standard way to handle this critical situation is to rebuild the complete data structure using two new hash functions. Fortunately, the occurrence of an endless loop is a rare event. More precisely, both variants of cuckoo hashing succeed with probability $1 - \mathcal{O}(1/m)$, conditioned that the load factor[(i)] is less than 0.5, see Devroye and Morin (2003), Drmota and Kutzelnigg (2009), and Pagh and Rodler (2004).

The analysis is hereby usually based on the assumption that the values of the hash functions form a sequence of independent uniform random numbers. This seems to impose strong conditions on the hash functions in use. However, numerical results obtained using quite simple hash functions are in good accordance with the results obtained by this model. Thus, we assume the same conditions on the hash functions to be satisfied in this paper too. A further discussion of this model, hash functions suitable for practical implementation, and additional references are given in Kutzelnigg (2009), but see also Dietzfelbinger and Schellbach (2009).

Although the success probability of cuckoo hashing tends to one under the conditions stated above, there is still a non negligible failure rate, even for large data structures. For instance, consider the outcome of one of the numerical experiments given in Table 8: There occurred 14355 errors among 10^7 constructions of data structures, each possessing in total 10^5 memory cells and a load factor $\alpha = 0.45$. Clearly, rebuilding a table requires high additional running time. Hence, cuckoo hashing is not attractive for applications where this behavior is not acceptable. To overcome this weak point, Kirsch et al. (2008) suggested to use a small additional memory, the so called stash. This stash is used to store items that could not be placed in the table itself. Interestingly, this modification drastically reduces the failure rate. More precisely, the authors showed that an additional memory of s items is sufficient to achieve a success probability of $1 - \mathcal{O}\left(m^{-s-1}\right)$ for the standard algorithm. However, the analysis is based on the usage of a modified insertion algorithm that requires additional steps to look for an admissible keys that can be placed in the stash. Details will be discussed in Section 4. We will show that the same result can be achieved without this search process, as it was already conjectured in Kirsch et al. (2008). Next, we prove that the same bound is valid for the simplified version of cuckoo hashing. Further, we demonstrate how detailed asymptotic expansions of the success rate can be obtained using a generating function approach.

(i) The load factor is defined as the quotient of the number of keys stored in the data structure and the total number of storage positions.

In particular, this can be used to show that the failure rate is in fact in $\Theta\left(m^{-s-1}\right)$. Finally, we support our analysis by numerical results.

2 The Cuckoo Graph

Our analysis is based on the cuckoo graph, a concept first developed in Devroye and Morin (2003). Hereby, standard cuckoo hashing without a stash is modeled using a bipartite multi graph. Each memory cell corresponds to a labeled node. Further, a node is colored "red" if and only if the according cell belongs to the first half of the memory slots. All other vertices are colored "blue". We further assume that the whole data structure contains $2m$ storage positions, each part numbered from 1 to m. We label the nodes accordingly. Next, each key is encoded by an edge that joins the two possible storage locations of that key. Additionally, the i-th inserted edge is labeled by i to encode the evolution. Note that the obtained graph is bipartite by construction.

Clearly, it is impossible for the algorithm to succeed if a component exists that has more edges than vertices. Thus, all components of the cuckoo graph must either be trees or unicyclic, i.e. contain either none or exactly one cycle. Interestingly, this condition is not only necessary, but also sufficient. More precisely, the key observation in Devroye and Morin (2003) is that cuckoo hashing succeeds if and only if all connected components of the cuckoo graph are either trees or unicyclic. Further, it is common to call the "bad" parts, i.e. connected components possessing more than one cycle, complex.

Concerning simplified cuckoo hashing, a similar modeling is possible, see Kutzelnigg (2010). In contrast to the bipartite graph described above, the one-table data structure is represented by a non-bipartite edge and node labeled directed graph. Edges are again used to represent keys and they still connect the two possible storage locations. However, it is necessary to use directed edges, to determine the primary storage position uniquely. This is closely related to the multi graph process described in Janson et al. (1993). Despite these differences in the definition of the graph model, simplified cuckoo hashing is again successful if and only if the shadow graph[(ii)] of its cuckoo graph does not contain a component with more than one cycle.

Additionally to the obvious implementation of the simplified algorithm that allows both storage positions of a key to be equal, there is a further method. Assume that the data structure contains $2m$ storage positions. For each key x, we assume that the hash value of the second hash function $h_2(x)$ is selected uniformly at random among the set $\{1, 2, \ldots, 2m\} \setminus h_1(x)$. This can be achieved by using a function $k(x)$ that maps x to a (pseudo) random number from 1 to $2m - 1$ and setting $h_2(x) = h_1(x) + k(x) \mod 2m$. As a small drawback, the evaluation of $h_2(x)$ requires an additional summation. On the other hand, each key now possesses in any way two different storage positions and the corresponding cuckoo graph does not contain self-loops. Consequently, we expect increased performance. In fact, this can be verified by a theoretical analysis, see Kutzelnigg (2010). However, note that the actual behavior is very similar, especially if tables are getting full.

[(ii)] Given a directed graph, we obtain its shadow graph by replacing each edge by an undirected one. Note that we consider multi graphs, thus multiple edges that occur in this process are retained.

3 Results

Theorem 1 *Consider a standard cuckoo hash data structure possessing two subtables of size m and holding $n = \lfloor 2\alpha m \rfloor$ keys, where $\alpha \in (0, 0.5)$ is fixed. Whenever an endless loop is detected, the last kicked-out element is placed in the stash of size $s \geq 1$. Then, the construction succeeds with probability*

$$1 - c(\alpha, s)\, m^{-s-1} + \mathcal{O}\left(m^{-s-2}\right). \tag{1}$$

Hereby, $c(\alpha, s) \neq 0$ depends on α and s, but not on m and it can be calculated explicitly with our method.

An essential difference of this result compared to (Kirsch et al., 2008, Theorem 1) is that we do not require additional steps to search for admissible keys. Further, our asymptotic result is much more precise.

The proof of Theorem 1 is split up into two parts. First, a proof that the described insertion strategy succeeds with probability $1 - \mathcal{O}\left(m^{-s-1}\right)$ is given in Section 4. These considerations are a refinement of the original analysis that can be found in Kirsch et al. (2008). The further proof is based on a generating function approach on the cuckoo graph and can be found in Section 7. In particular, we prove that $c(\alpha, s) \neq 0$ holds. Moreover, we present a method to compute this coefficient, and we have implemented it with Maple. However, note that the calculations are limited by the available memory of the machine that executed the computer algebra system. Using a workstation equipped with 12GB RAM, we were successful in solving the problem for $s \in \{0, 1, 2\}$.

Note that during a successful insertion procedure, each memory slot is visited at most twice. Thus, an insertion procedure in a table previously holding i keys is clearly unsuccessful if it takes more than $2i + 1$ kick-outs. However, we can usually do much better and stop earlier, see Kutzelnigg (2009) for corresponding numerical data. On the other hand, there is a risk that we stop procedures that would have been successful. The probability can be bounded as follows: For each k, there exists a sufficiently large number $\beta = \beta(k)$ that does not depend on m, such that the probability that there is an insertion that takes more than $\beta \log m$ steps is $\mathcal{O}\left(m^{-k}\right)$, see (Kirsch et al., 2008, Lemma 4 and (1)). Hence, (1) still holds if an insertion procedure is stopped and declared unsuccessful after $\beta \log m$ kick-outs performed, provided that β is chosen sufficiently large.

Furthermore, we can analyze the simplified version by similar methods, see Sections 5 and 7 for details. Note that we distinguish between the straightforward implementation, and the version without self-loops as described in the previous section.

Theorem 2 *Consider a simplified cuckoo hash table possessing $2m$ memory cells and holding $n = \lfloor 2\alpha m \rfloor$ keys, where $\alpha \in (0, 0.5)$ is fixed. Suppose that h_2 is implemented such that $h_2(x) \neq h_1(x)$ holds for all keys x. Whenever an endless loop is detected, the last kicked-out element is placed in the stash of size $s \geq 1$. Then, the construction succeeds with probability*

$$1 - \overline{c}(\alpha, s)\, m^{-s-1} + \mathcal{O}\left(m^{-s-2}\right).$$

Hereby, $\overline{c}(\alpha, s) \neq 0$ depends on α and s, but not on m and it can be calculated explicitly with our method.

Similar to the considerations given above, it is again possible to stop presumably unsuccessful insertion procedures early.

Until now, we have successfully calculated the numbers $\overline{c}(\alpha, s)$ for all $s \leq 8$. Unfortunately, Theorem 2 does not cover the situation where self-loops are not excluded. However, we can analyze it with our generating function approach and yield:

Theorem 3 *Suppose that the conditions of Theorem 2 are fulfilled, except that the data structure is implemented in the way such that self-loops may occur. Further, let $s \in \{0, 1, \dots, 8\}$. Then, the probability that the construction of a simplified cuckoo hash table succeeds equals*

$$1 - \hat{c}(\alpha, s)\, m^{-s-1} + \mathcal{O}\left(m^{-s-2}\right).$$

Hereby, $\hat{c}(\alpha, s) \neq 0$ depends on α and s, but not on m and it can be calculated explicitly with our method.

The proof of this result can be found in Section 7.

4 Insertion

The results given in this section hold for both types of cuckoo graphs. However note that the original paper of Kirsch et al. (2008) considered the bipartite version only.

4.1 The insertion procedure of Kirsch et al. (2008)

Once an endless loop is discovered, it is required to search for an admissible key to be put in the stash, such that the remaining keys can be placed in the original hash table. Clearly, a key is acceptable if and only if the cuckoo graph does not possess a complex component after removing the coressponding edge. We continue with an observation stated in Kirsch et al. (2008), and we are using the same notation as in that paper: Given the cuckoo graph G, we denote the total number of edges that closed a cycle when inserted by $f(G)$. Further, the number of the connected components containing at least one cycle is denoted by $T(G)$. Consequently, the number of elements that could not be stored in the table itself and have to be placed in the stash, equals $f(G) - T(G)$. Moreover, we continue describing the modified insertion procedure used in Kirsch et al. (2008): Whenever an endless loop is detected, the last insertion step created a component possessing two cycles. We proceed searching for an edge of that component that is contained in one of this two cycles. This is done by counting how often a memory cell is accessed during an insertion and thus, it is quite computational expensive resp. memory consuming. Next we delete the selected edge from the graph and put the corresponding key into the stash. Note that an edge that closes a cycle on insertion in the original graph, still closes a cycle if we have removed some edges as described above. Hence, we have found a way to remove exactly $f(G) - T(G)$ edges, such that all conflicts are resolved. Clearly, removing less edges can not repair all problems and hence the solution has minimal cardinality.

4.2 Insertion without search for an admissible key

However, removing an edge that belongs to a cycle is not the only possible way to decrease $f(G) - T(G)$. We can alternatively choose the edge such that $T(G)$ increases. This corresponds to breaking up a component in two parts, but both of that new components must not be trees. We proceed using the following definition:

Definition 1 *Consider a connected graph H. The excess $e(H)$ is given by*

$$e(H) = \#\,\mathrm{edges}(H) - \#\,\mathrm{nodes}(H).$$

Furthermore, for a graph G, we define

$$\tilde{e}(G) = \sum_H \max(e(H), 0) = f(G) - T(G),$$

where the sum is taken over all components H of G.

The insertion procedure is modified as follows: As long as no irresolvable conflict occurs in a basic cuckoo hash data structure, we perform the insertions using the original algorithm of Pagh and Rodler (2004) that is described in Section 1. However, each time we detect an endless loop, we put the last kicked-out key in the stash, and delete the corresponding edge. More precisely, we even may select that key arbitrarily among the set of all keys that have been kicked out in the current insertion procedure, but it is of course convenient to choose the last evicted key. We then proceed in the usual way to insert further keys, until the next problem is discovered. Thus, we avoid a complicated cycle detection mechanism.

The remainder of this section is used to proof that this strategy is alway successful. First, we say that an edge is disturbing, if its insertion increases $\tilde{e}(G)$. Recall that we are considering labeled and thus ordered edges. Since $\tilde{e}(G)$ cannot decrease by inserting an additional edge[(iii)], $\tilde{e}(G)$ equals the number of disturbing edges. However, it is possible to decrease $\tilde{e}(G)$ by removing certain edges:

Lemma 1 *Removing an edge g of G decreases $\tilde{e}(G)$ if and only if the removal of that edge does not create a new tree.*

Proof: Denote the component that contains g by H. First, if g is contained in a cycle, $e(H\setminus g) = e(H)-1$ holds. Thus $\tilde{e}$ decreases by one, except if the excess of H equals 0. But in the latter case, $H \setminus g$ is a new tree. Second, if g is not contained in a cycle, it is a bridge connecting H_1 and H_2. Hence, the relation $e(H) = e(H_1) + e(H_2) + 1$ is satisfied. If both H_1 and H_2 are trees, H is also a tree and none of these components influences the calculation of $\tilde{e}$. If only one component is a tree, say H_2, $e(H) = e(H_1)$ holds and $\tilde{e}$ remains again unchanged. However, if both H_1 and H_2 posses nonnegative excess, we obtain $\tilde{e}(G \setminus g) = \tilde{e}(G) - e(H) + e(H_1) + e(H_2) = \tilde{e}(G) - 1$. □

Next, we consider the influence of removed edges on further disturbing edges:

Lemma 2 *Let G' denote a graph arising from G, where we have already removed $\tilde{e}(G)$ edges such that $\tilde{e}(G') = 0$ holds. Assume that a new edge g is disturbing in G'. Then, it is disturbing in G too.*

Proof: We consider two different situations in G', see Figure 1. First, suppose that g is a bridge in G' that connects H'_1 and H'_2. By similar reasoning as in the proof of Lemma 1, both of this components cannot

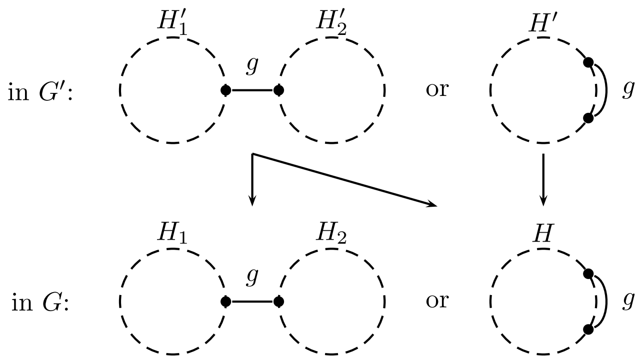

Fig. 1: A disturbing edge g. We distinguish whether g is a bridge in G or not.

(iii) Note that the number of nodes is fixed, and hence remains unchanged.

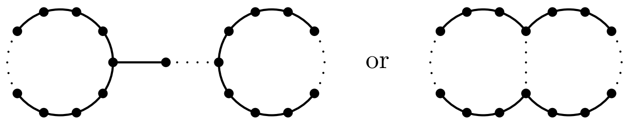

Fig. 2: The two possible types of bicyclic components.

be trees, because otherwise g would not be a disturbing edge. Thus, $e(H_1') \geq 0$ and $e(H_2') \geq 0$ hold. Assume that g is a bridge in G that connects components $H_1 \supseteq H_1'$ and $H_2 \supseteq H_2'$. We conclude that $e(H_1) \geq e(H_1') \geq 0$ and $e(H_2) \geq e(H_2') \geq 0$ are fulfilled. Hence g is disturbing in G. The proof of the two further cases follows by a similar argument as above. □

Lemma 3 *Assume that we always perform the following steps whenever the insertion procedure has entered an endless loop: We stop at an arbitrarily selected moment, put the current homeless key in the stash of potential unlimited capacity, and delete the corresponding edge. By doing so, the stash holds $\tilde{e}(G)$ keys finally, where G denotes the original cuckoo graph without deleted edges.*

Proof: Consider the graph G' that encodes the current state of the data structure. The algorithm enters an endless loop whenever a component containing two cycles is created. This can either happen by closing a further cycle in an unicyclic component or by connecting two different unicyclic components. Hence, the new edge is disturbing for G'. There exist two different typical situations, depicted in Figure 2. Each of the nodes can be considered as root of an additional attached tree, not depicted in the figure. Observe that the last inserted edge must be contained in the drawn part. None of the keys belonging to one of the depicted edges has a possible storage position in these potentially attached trees. Hence the presence of this appendix has no influence on the current insertion. In particular, none of these keys can ever become kicked-out in the current insertion operation. Note that arbitrarily selection and deletion of one the drawn edge decreases $\tilde{e}$, because of Lemma 1.

Note that the frequency of the occurrence of a disturbing edge is bounded by $\tilde{e}(G)$, due to Lemma 2. Hence, we place at most $\tilde{e}(G)$ keys in the stash. On the other hand, this is the minimum number of elements that have to be stored outside the table. □

To prove Theorem 1, it hence remains to show that $\mathbb{P}(\tilde{e}(G) \geq s+1)$ has asymptotic expansion (1). This is done in Section 7. Due to (Kirsch et al., 2008, (2)), the relation $\mathbb{P}(\tilde{e}(G) \geq s+1) = \mathbb{P}(f(G) - T(G) \geq s+1) = \mathcal{O}(m^{-s-1})$ is already known. We just want to correct a minor mistake. When an insertion hits a cyclic component, several (and not only one) vertices might be visited twice. In particular, $k+1$ is not a valid upper bound for the number of steps that an insertion requires in a component of size k, as claimed in (Kirsch et al., 2008, Lemma 1). However, $2k$ is sufficient bound, see Devroye and Morin (2003) or Kutzelnigg (2009) for further details.

5 Simplified Cuckoo Hashing

Instead of segmenting the available memory, this modified version grants both hash functions access to the whole table. This variant was first mentioned in Pagh and Rodler (2004), a detailed analysis can be found in Kutzelnigg (2009, 2010). In particular, the simplified algorithm offers improved performance of search and insertion operations.

In this section, we prove that the data structure of Theorem 2 succeeds with probability $1-\mathcal{O}(m^{-s-1})$. First, recall that all lemmata given in Section 4 did not require a bipartite structure of G. Hence, $\tilde{e}(G)$ still equals the number of keys that can not be stored in the table itself. Recall that we consider the implementation without self-loops only. This is based on the fact that it is possible to adopt the analysis of Kirsch et al. (2008) for this version. Nevertheless, because of the results given in Section 7, we conjecture that the same bound holds for the other implementation too. Our proof is based on a stochastic dominance argument, and we repeat resp. adopt the following definitions and results from Kirsch et al. (2008): For a distribution D, let $\mathcal{G}(2m, D)$ denote the distribution over graphs with $2m$ nodes, obtained by sampling $l \sim D$ and inserting l edges independently. Hereby, start point and end point of each edge are selected uniformly and independently, except that no self loops are allowed. Clearly, the cuckoo graph of the simplified variant has distribution $\mathcal{G}(2m, D)$ when D is concentrated at n.

Given two graphs G and G' possessing the same set of vertices, we say that $G \geq G'$ holds if each edge of G is also contained in G'. Additionally, we generalize this relation on t-tuples of graphs by applying it component-by-component, i.e. $(G_1, \ldots, G_t) \geq (G'_1, \ldots, G'_t)$ holds if $G_i \geq G'_i$ is fulfilled for all i.

Definition 2 *Let μ and ν be two probability measures over t-tuples of graphs with common vertex set. Then, μ stochastically dominates ν, in short notation $\mu \succeq \nu$, if $\mathbb{E}_\mu(g(G)) \geq \mathbb{E}_\nu(g(G))$ holds for all non-decreasing functions g.*

Further, let $\mathrm{Po}(\lambda)$ denote the Poisson distribution with parameter λ.

Lemma 4 *Assume that $\lambda > 0$ is fixed. Then, the conditional distribution of $G \sim \mathcal{G}(2m, \mathrm{Po}(\lambda))$, conditioned on the property that G has at least n edges stochastically dominates $\mathcal{G}(2m, n)$.*

Proof: The proof follows the lines of (Kirsch et al., 2008, Lemma 3). Obviously, the conditional distribution of G under the assumption that there are exactly n edges is $\mathcal{G}(2m, n)$. Since $k_1 > k_2$ implies $\mathcal{G}(2m, k_1) \succeq \mathcal{G}(2m, k_2)$, the result follows. □

It is easy to see that the parameter λ can bee chosen such that the probability that $\mathcal{G}(2m, \mathrm{Po}(\lambda))$ has less than n edges is exponentially small:

Lemma 5 (Kirsch et al. (2008)) *Let $\varepsilon' > 0$ be fixed and define $\lambda = (1+\varepsilon')n$. Then we yield*

$$\mathbb{P}(\mathrm{Po}(\lambda) < n) = e^{-\Omega(n)}.$$

Thus, it remains to show that for $G \sim \mathcal{G}(2m, \mathrm{Po}(\lambda))$ the relation

$$\mathbb{P}\big(\tilde{e}(G) \geq s\big) = \mathcal{O}(m^{-s}) \tag{2}$$

holds. Given a vertex v of $G \sim \mathcal{G}(2m, \mathrm{Po}(\lambda))$, let C_v denote the connected component that contains v. Furthermore, let B_v denote the number of edges that belong to C_v that closed a circle on insertion.

Lemma 6 *Assume that $\varepsilon' < 1/(2\alpha) - 1$ and t,k,n ≥ 1 hold. Then, we get*

$$\mathbb{P}(B_v \geq t \,|\, |C_v| = k) \leq \left(\frac{3e^5k^3}{2m}\right)^t.$$

Proof: The condition on ε' ensures that $\lambda = (1+\varepsilon')n$ is selected such that $\lambda/m < 1$ holds. As in the proof of (Kirsch et al., 2008, Lemma 5), we consider a breath first search starting at v. We bound the number of cycles that are created by visiting a new node by a Poisson random variable in each step. The only difference is that a single edge now corresponds to a Poisson random variable with parameter $\lambda/\binom{2m}{m}$, instead of λ/m^2. Thus, we can apply (Kirsch et al., 2008, Lemma 6) with m replaced by $2m$ and yield the claimed result. $\square$

By the same reasoning as (Devroye and Morin, 2003, Lemma 1), we obtain the following result for the size of a component:

Lemma 7

$$\mathbb{P}(|C_v| > k) \le \mathbb{P}(\mathrm{Bin}(nk, 1/(2m-1)) \ge k)$$

Lemmata 6 and 7 together with Chernoff's bound are sufficient to prove (2), see Kirsch et al. (2008).

6 Enumerating complex graphs

In this section, we review resp. establish the generating functions that enable us to count the number of cuckoo graphs of certain type resp. asymptotic approximations of these numbers. First, let us consider node and edge labeled usual, but directed graphs. Let $\mathcal{F}$ denote a family of such graphs. Then, the corresponding bivariate generating function is the formal power series

$$F(x,v) = \sum_{\mathcal{G}\in\mathcal{F}} \frac{x^{m(\mathcal{G})}}{m(\mathcal{G})!} \frac{v^{n(\mathcal{G})}}{n(\mathcal{G})!},$$

where $m(\mathcal{G})$ denotes the number of nodes and $n(\mathcal{G})$ the number of edges of $\mathcal{G}$. However, it is usually simpler to consider an univariant generating function first. For instance, node labeled rooted trees are enumerated by the well known tree function $t(x)$ satisfying $t(x) = xe^{t(x)}$, see Flajolet and Sedgewick (2009). Since a tree possesses exactly one node more than edges, the corresponding bivariate generating function $t(x,v)$ satisfies

$$t(x,v) = \frac{t(2xv)}{2v}.$$

We thus slightly abuse notation by denoting univariate and bivariate generating function by the same letter, however the correct interpretation should be clear from the context.

Similarly, we define trivariate generating functions enumerating bipartite graphs:

$$F(x,y,v) = \sum_{\mathcal{G}\in\mathcal{F}} \frac{x^{m_1(\mathcal{G})}}{m_1(\mathcal{G})!} \frac{y^{m_2(\mathcal{G})}}{m_2(\mathcal{G})!} \frac{v^{n(\mathcal{G})}}{n(\mathcal{G})!},$$

Again, we avoid using the edge-counting variable v whenever possible.

Further, we make use of the notation $[x^m]A(x)$ to extract the m-th coefficient of a power series $A(x)$ that means

$$[x^m]A(x) = [x^m]\sum_{k\ge 0} a_m x^m = a_m.$$

6.1 Usual graphs

Additionally to the already mentioned function $t(x)$ counting rooted trees, the function $\tilde{t}(x)$ counting unrooted trees is well-known too. In particular, the relation

$$\tilde{t}(x) = t(x) - \frac{1}{2}t(x)^2,$$

holds, see Flajolet and Sedgewick (2009). Using these functions, we describe the generating functions of graphs with excess $r \geq 0$. Since the number of nodes is uniquely determined for all these types of graphs, we can concentrate on counting nodes. Thus univariate generating functions are sufficient, because we get the bivariate function afterwards by replacing x with $2xv$ and multiplying the function by an additional factor of $(2v)^r$. However, this can only be done if self-loops and multiple edges are compensated in the original construction. We follow the approach of Janson et al. (1993) and assign a graph with adjacency matrix $A = (a_{ij})_{ij}$ the compensation factor

$$\kappa(A) = \left(\prod_i 2^{a_{ii}} \sum_{j \geq i} a_{ij}!\right)^{-1}.$$

In particular, the following results hold for the general multi graph model:

Lemma 8 (Janson et al. (1993)) *Assume that our graph might contain self-loops. Then, the generating function $C(x)$ of a connected graph with exactly one cycle is given by*

$$C(x) = \sum_{k \geq 1} \frac{1}{2k} t(x)^k = \log \frac{1}{\sqrt{1 - t(x)}}.$$

Further, let $E_r(x)$ denote the generating function of graphs consisting of complex components only and having excess r, i.e. exactly r more edges than vertices. Then the relation

$$E_r(x) = \sum_{d=0}^{2r} \frac{e_{rd}\, t(x)^{2r-d}}{(1 - t(x))^{3r-d}}$$

holds. Hereby, the constants e_{rd} are given as

$$\frac{(6r - 2d)! P_d(r)}{2^{5r} 3^{2r-d} (3r - d)!(2r - d)!}$$

where $P_d(r)$ is further defined by $P_d(r) = [x^d]F(x)$ and

$$F(x) = \frac{6}{(4x)^3}\left(e^{4x} - \frac{(4x)^2}{2} - 4x - 1\right).$$

Since we also consider the situation where each key possesses two different storage positions, we also require the generating functions counting graphs without self-loops.

Definition 3 *Let ϑ_x denote the differential operator $x\frac{\partial}{\partial x}$ that corresponds to marking a node.*

Lemma 9 *Assume that we consider a multi graph without self-loops. Then, the generating function $\overline{C}(x)$ of a connected graph with exactly one cycle is given by*

$$\overline{C}(x) = \sum_{k\geq 2} \frac{1}{2k} t(x)^k = \log \frac{1}{\sqrt{1-t(x)}} - \frac{1}{2} t(x).$$

Further, let $\overline{E}_r(x)$ denote the generating function of graphs consisting of complex components only and having excess r, i.e. exactly r more edges than vertices. These functions satisfy the differential recurrence

$$(r + (1-t)\vartheta_x)\,\overline{E}_r = \frac{1}{2} e^{-\overline{C}} \left(\vartheta_x^2 - \vartheta_x\right) e^{\overline{C}} \overline{E}_{r-1}. \tag{3}$$

Moreover, $E_0 = 1$ holds, since only the empty graph is complex and has excess 0.

Note that the previously mentioned results are not explicitly given in Janson et al. (1993). However, the case where both, self-loops and multiple edges are forbidden, is considered in that paper. Thus, the proof of Lemma 9 follows immediately by combining these methods and results. Finally, using the differential recursion (3) and a computer algebra system, it is easy to obtain all generating functions that are required in the next section.

6.2 Bipartite graphs

In contrast to usual graphs, only few results concerning bipartite graphs are known in the literature. However, trees and unicyclic components have been studied recently, see also Gimenez et al. (2005):

Lemma 10 (Drmota and Kutzelnigg (2009)) *The generating functions $t_1(x,y)$ resp. $t_2(x,y)$ of rooted bipartite trees possessing a root node of first resp. second type and the generating function of unrooted bipartite trees $\tilde{t}(x,y)$ are given by*

$$t_1(x,y) = x e^{t_2(x,y)}, \quad t_2(x,y) = y e^{t_1(x,y)}$$

and

$$\tilde{t}(x,y) = t_1(x,y) + t_2(x,y) - t_1(x,y) t_2(x,y).$$

The partial derivatives of the functions $\tilde{t}(x,y)$ and $t_1(x,y)$ are given by

$$\frac{\partial}{\partial x}\tilde{t}(x,y) = \frac{t_1(x,y)}{x}, \qquad \frac{\partial}{\partial y}\tilde{t}(x,y) = \frac{t_2(x,y)}{y},$$

and

$$\frac{\partial}{\partial x} t_1(x,y) = \frac{t_1(x,y)}{x(1-t_1(x,y)t_2(x,y))}, \qquad \frac{\partial}{\partial y} t_1(x,y) = \frac{t_1(x,y)\, t_2(x,y)}{y(1-t_1(x,y)t_2(x,y))}.$$

Further, the generating function of a connected bipartite graph with exactly one cycle is given by

$$C(x,y) = \sum_{k\geq 1} \frac{1}{2k} t_1(x,y)^k t_2(x,y)^k = \frac{1}{2} \log \frac{1}{1 - t_1(x,y) t_2(x,y)}.$$

Concerning components with positive excess, we proceed as in Janson et al. (1993) and adopt the calculation to the present situation. Thereby, we make use of the following shortened notation:

Definition 4 *Let ϑ_x denote the differential operator $x\frac{\partial}{\partial x}$ that corresponds to marking a vertex of first kind. Similarly, we define the operators ϑ_y and ϑ_v for marking a node of second kind resp. an edge.*

Further, we obtain the following result:

Lemma 11 *Let $E = E(x, y, v)$ denote the generating function for the complex part of a bipartite graph that means all its components have positive excess. Then, the following equation holds:*

$$\frac{1}{v}\vartheta_v E = (\vartheta_x\vartheta_y C)E + (\vartheta_x C)(\vartheta_y C)E \\ + (\vartheta_x\tilde{t})(\vartheta_y E) + (\vartheta_x E)(\vartheta_y\tilde{t}) + (\vartheta_x C)(\vartheta_y E) + (\vartheta_x E)(\vartheta_y C) + \vartheta_x\vartheta_y E \quad (4)$$

Proof: Obviously, the left hand side of our equation represents all complex bipartite graphs having a marked edge with the edge count decreased by one. Thus, the right hand side should yield all ways how the complex part can grow by one edge. In particular, the terms correspond to the following operations from left to right:

- The new edge connects two edges of an unicyclic component and thus closes a further loop in it.
- It might also join two different unicyclic components, and hence a bicyclic component arises.
- Furthermore, the new edge might attach a tree to an already complex component. There are two distinct situations, the node of first kind can belong to each of the two involved components.
- Similar to the previous case, but we select an unicyclic component instead of a tree.
- Finally, we may connect two edges belonging to the complex part.

Additionally it is straightforward to verify (4) adopting the calculations of Janson et al. (1993), but we skip this tedious and technical alternative proof. □

To solve the differential equation (4), we transform it into a recursion formula:

Lemma 12 *Let $E_r(x, y)$ denote the generating function of bipartite graphs consisting of complex components only and having excess r. These functions satisfy the partial differential recurrence*

$$\left(r + (1 - t_2)\vartheta_x + (1 - t_1)\vartheta_y\right) E_r = e^{-C}\vartheta_x\vartheta_y e^C E_{r-1}.$$

Moreover, $E_0 = 1$ holds, since only the empty graph is complex and has excess 0.

Proof: We start rewriting (4). Using Lemma 10, we obtain

$$\vartheta_x\tilde{t}(x, y, v) = \frac{t_1(xv, yv)}{v}, \quad \vartheta_y\tilde{t}(x, y, v) = \frac{t_2(xv, yv)}{v}.$$

Furthermore, it is straightforward to verify that

$$(\vartheta_x\vartheta_y C)E + (\vartheta_x C)(\vartheta_y C)E + (\vartheta_x C)(\vartheta_y E) + (\vartheta_x E)(\vartheta_y C) + \vartheta_x\vartheta_y E = e^{-C}\vartheta_x\vartheta_y e^C E$$

holds. Thus we get

$$\frac{1}{v}(\vartheta_v - t_1(xv,yv)\vartheta_y - t_2(xv,yv)\vartheta_x)E(x,y,v) = e^{-C(x,y,v)}\vartheta_x\vartheta_y e^{C(x,y,v)}E(x,y,v). \tag{5}$$

Next we partition $E(x,y,v)$ into summands of equal excess. Thus we write

$$E(x,y,v) = \sum_r E_r(x,y,v) = \sum_r v^r E_r(xv,yv).$$

Further, we immediately yield

$$\vartheta_x E(x,y,v) = \sum_r v^r x \frac{\partial}{\partial x} E_r(xv,yv) = \sum_r v^r \left(\vartheta_x E_r\right)(xv,yv), \tag{6}$$

and similarly

$$\vartheta_v E(x,y,v) = \sum_r \big(r v^r E_r(xv,yv) + v^r \big((\vartheta_x + \vartheta_y)E_r\big)(xv,yv)\big) \tag{7}$$

holds. Hereby $(\vartheta_x E_r)(xv,yv)$ means $\vartheta_x E_r(x,y)$ with x and y subsequently replaced by xv and yv. Finally, we plug in (6) and (7) into (5), equate the coefficients of v^{r-1} on both sides and then set $v = 1$. Thus we obtain

$$\left(r + (1 - t_2(x,y))\vartheta_x + (1 - t_1(x,y))\vartheta_y\right) E_r(x,y) = e^{-C(x,y)}\vartheta_x\vartheta_y e^{C(x,y)}E_{r-1}(x,y),$$

what completes the proof. □

Solving the recursion of Lemma 12 in general seems to be out of reach, but using a computer algebra system, it is quite easy to get some results. In particular, the solutions exhibit a certain pattern, hence it is possible to write down an ansatz and compute the coefficients. Thus, we get for instance

$$E_1 = \frac{t_1t_2(4 + 3t_2 + 3t_1 + 6t_1t_2 + 2t_1t_2^2 + 2t_1^2t_2)}{24(1 - t_1t_2)^3},$$

and

$$\begin{aligned} E_2 = \frac{t_1t_2}{1152(1-t_1t_2)^6}\big(&720t_1^2t_2 + 24t_1^2 + 652t_1^3t_2^2 + 72t_1^4t_2^3 + 156t_1^4t_2^2 + 156t_2^4t_1^2 \\ &+ 48 + 24t_2^2 + 201t_1^3t_2 + 4t_2^5t_1^3 + 688t_1t_2 + 4t_1^5t_2^3 + 201t_1t_2^3 + 8t_1^4t_2^4 \\ &+ 348t_1^3t_2^3 + 72t_2^4t_1^3 + 652t_1^2t_2^3 + 1218t_1^2t_2^2 + 720t_1t_2^2 + 96t_2 + 96t_1\big). \end{aligned}$$

7 Detailed Asymptotic Expansions

In this section, we present a generating function approach that enables us to calculate detailed asymptotic expansions of the number of cuckoo graphs satisfying $\tilde{e}(G) \le s$. Together with the number of all cuckoo graphs, we further obtain the percentage of graphs that fulfill the very same property. Hence, the proofs follow the same idea as the analysis of the success probability given in Drmota and Kutzelnigg (2009) and Kutzelnigg (2009).

7.1 Usual graphs

We concentrate on the case where self-loops are allowed. The second case can be treated in the same way, by just exchanging the generating functions. Thus, we obtain a similar result, however the non-zero coefficients of the asymptotic expansion are different.

First, it is straightforward to count all directed node and edge labeled multi graphs possessing $2m$ nodes an n edges:

$$\#G_{2m,n} = \left[\frac{v^n}{n!}\right] (e^v)^{2m} \left(e^{2v}\right)^{\binom{2m}{2}} = (4m^2)^n.$$

Second, we determine $\#G^r_{2m,n}$, the number of all graphs containing a complex part with r more edges than nodes, using the generating functions given in the previous section. In particular, each of these graphs contains $2m-n+r$ (unrooted) tree components. This is easy to see, because of the fact that the insertion of an edge reduces the number of tree components by one, except if it increases the excess of the complex part. Next, the graph contains a (possibly empty) set of unicyclic components. Hence we infer by elementary combinatorial constructions (see e.g. Flajolet and Sedgewick (2009)):

$$\#G^r_{2m,n} = \frac{(2m)!\,n!}{2^{2m-n}}[x^{2m}]\frac{\tilde{t}(2x)^{2m-n+r}}{(2m-n+r)!}e^{C(2x)}E_r(2x) = \frac{2^n(2m)!\,n!\,\left[x^{2m}\right]\tilde{t}(x)^{2m-n+r}e^{C(x)}E_r(x)}{(2m-n+r)!}. \tag{8}$$

We use Cauchy's Formula to obtain an integral representation of that formula. This integral can be asymptotically evaluated using the saddle point method, see Lemma 13 that can be found in the appendix. For technical reasons, we define the ratio

$$\varepsilon = 1 - \frac{n}{m} = 1 - \frac{\lfloor 2\alpha m\rfloor}{m} = 1 - 2\alpha + \mathcal{O}(m^{-1}).$$

Then, it turns out that the saddle point is given by $x_0 = (1-\varepsilon)e^{\varepsilon-1}$. In particular, the result for the special case $r = 0$ that corresponds to an empty stash, is given in Drmota and Kutzelnigg (2009) and Kutzelnigg (2009). However, in the present situation, it is not sufficient to provide the second order term of the expansion only, we require several further terms. The calculation of these asymptotic expansions has been done with Maple in a semi-automatic way. In particular, we obtain

$$\#G^r_{2m,n} = \#G_{2m,n}\left(\frac{c^r_r(\varepsilon)}{m^r} + \frac{c^r_{r+1}(\varepsilon)}{m^{r+1}} + \frac{c^r_{r+2}(\varepsilon)}{m^{r+2}} + \dots\right), \tag{9}$$

where the $c^r_i(\varepsilon)$ depend on ε, but not on m. The first non-zero coefficient is given by

$$c^r_r(\varepsilon) = \frac{E(x_0)t(x_0)^r}{2^r}, \tag{10}$$

and in particular $c^0_0(\varepsilon) = 1$ holds.

We conclude that the fraction of cuckoo graphs that possesses a cyclic part of excess r is given by $\#G^r_{2m,n}/\#G_{2m,n}$. Then, the percentage of graphs possessing a single bicyclic component of excess one equals

$$\frac{\#G^1_{2m,n}}{\#G_{2m,n}} = \frac{c^1_1(\varepsilon)}{m} + \frac{c^1_2(\varepsilon)}{m^2} + \mathcal{O}\left(\frac{1}{m^3}\right)$$
$$= \frac{(1-\varepsilon)^2(5-2\varepsilon)}{48\varepsilon^3}\frac{1}{m} + \frac{(1-\varepsilon)(4\varepsilon^5 - 104\varepsilon^4 + 481\varepsilon^3 - 727\varepsilon^2 + 1127\varepsilon - 925)}{2304\varepsilon^6}\frac{1}{m^2} + \mathcal{O}\left(\frac{1}{m^3}\right).$$

Furthermore, the probability that $\tilde{e}(G)$ is less or equal s for a randomly selected graph G is given by

$$\mathbb{P}(\tilde{e}(G) \le s) = \sum_{r=0}^{s} \frac{\#G^r_{2m,n}}{\#G_{2m,n}}$$
$$= c^0_0(\varepsilon) + \sum_{r=0}^{1}\frac{c^r_1(\varepsilon)}{m} + \sum_{r=0}^{2}\frac{c^r_2(\varepsilon)}{m^2} + \cdots \sum_{r=0}^{s}\frac{c^r_s(\varepsilon)}{m^s} + \sum_{r=0}^{s}\frac{c^r_{s+1}(\varepsilon)}{m^{s+1}} + \mathcal{O}\left(\frac{1}{m^{s+2}}\right).$$

Moreover, for $0 < i \le 8$ we verified

$$\sum_{r=0}^{i} c^r_i(\varepsilon) = 0,$$

using our Maple worksheets[(iv)]. Thus, we yield for $s \le 8$

$$\mathbb{P}(\tilde{e}(G) \le s) = 1 + \sum_{r=0}^{s}\frac{c^r_{s+1}(\varepsilon)}{m^{s+1}} + \mathcal{O}\left(\frac{1}{m^{s+2}}\right) = 1 - \frac{c^{s+1}_{s+1}(\varepsilon)}{m^{s+1}} + \mathcal{O}\left(\frac{1}{m^{s+2}}\right).$$

In particular,

$$\mathbb{P}(\tilde{e}(G) \le 1) = 1 + \sum_{r=0}^{2}\frac{c^r_2(\varepsilon)}{m^2}$$
$$= 1 - \frac{(1-\varepsilon)(4\varepsilon^5 - 32\varepsilon^4 + 97\varepsilon^3 + 149\varepsilon^2 + 959\varepsilon - 1465)}{4608\varepsilon^6}\frac{1}{m^2} + \mathcal{O}\left(\frac{1}{m^3}\right)$$

holds. Further, we can replace ε by $1 - 2\alpha + \mathcal{O}(m^{-1})$. We obtain similar results for all other s, satisfying $s \le 8$, what completes the proof of Theorem 3.

Finally, we prove that the failure rate of a data structure possessing a stash of size s is in fact $\Theta(m^{-s-1})$. That is, we show that $\overline{c}(\alpha, s)$ as defined in Theorem 2, is not equal to zero. To do so, we consider a complex component of excess $s+1$, consisting of exactly two nodes connected by $s+3$ edges. Clearly, the bivariate resp. univariate generating function b of such a component is given by

$$b(x,v) = \frac{x^2 v^{s+3}}{(s+3)!} \quad \text{resp.} \quad b(x) = \frac{x^2}{(s+3)!}.$$

(iv) Note that we conjecture a similar result for all further i. Moreover, for the situation of Theorem 2, this is implied by (2).

We further consider (8) once more, but we replace E by b and repeat the application of the saddle point method. Similarly to (10), we obtain that the probability that exactly one such component occurs in the cuckoo graph, while all other components are either trees or unicyclic, is given by

$$\frac{b(x_0)t(x_0)^{s+1}}{2^{s+1}m^{s+1}} \neq 0.$$

Since the algorithm fails in the current situation, the proof of Theorem 2 is completed.

7.2 Bipartite Graphs

In general, this proof follows the same idea as the previous one. However, things are a bit more complicated because of the bivariate generating functions that capture the bipartite structure of the cuckoo graph. Again, we start counting all graphs without restrictions to the type of their components. Let $G_{m_1,m_2,n}$ denote the set of all vertex and edge labeled bipartite multi graphs (V_1, V_2, E) with $|V_1| = m_1$, $|V_2| = m_2$, and $|E| = n$. By definition, it is obvious that the number of all graphs having m nodes of each type and n edges is given by

$$\#G_{m,m,n} = m^{2n}.$$

We proceed counting all bipartite graphs possessing a complex part of excess r. As in the univariate situation, such a graph further contains $2m - n + r$ unrooted tree components, and a (possibly empty) set of unicyclic components:

$$\#G^r_{m,m,n} = (m!)^2 n! [x^m y^m] \frac{\tilde{t}(x,y)^{2m-n+r}}{(2m+n-r)!} e^{C(x,y)} E_r(x,y).$$

An asymptotic expansion can be derived applying a double saddle point method, see Lemma 14 and Drmota and Kutzelnigg (2009). The saddle point is given by $x_0 = y_0 = \frac{n}{m} e^{-n/m}$. All calculations have again been done with Maple and we obtain an expansion as we had in (9), satisfying the same properties but the non-zero coefficients are different. Finally, the probability that the construction of a data structure requires a stash of at most s items is given by

$$\mathbb{P}(\tilde{e}(G) \le s) = \sum_{r=0}^{s} \frac{\#G^r_{m,m,n}}{\#G_{m,m,n}} = 1 - \Theta\left(m^{-s-1}\right),$$

and we yield the claimed results. In particular, setting again $\varepsilon = 1 - n/m$, we get

$$\mathbb{P}(\tilde{e}(G) \le 1) = 1 - \frac{(\varepsilon-1)^4(4\varepsilon^6 - 52\varepsilon^5 + 305\varepsilon^4 - 868\varepsilon^3 + 1358\varepsilon^2 - 1120\varepsilon + 385)}{288(-2+\varepsilon)^4\varepsilon^6} \frac{1}{m^2} + \mathcal{O}\left(\frac{1}{m^3}\right).$$

Finally, we use the same approach as for the usual graph to construct a cuckoo graph that requires a stash of size $s + 1$ and occurs with probability $\Theta(m^{-s-1})$.

8 Experimental Results

Finally, we provide some numerical results for both variants of cuckoo hashing. Thereby, we use a pseudo random generator to obtain the hash values. The stash is implemented as a linked list providing potential unlimited additional memory. Hence, the construction can never fail, but unsuccessful search operations require the inspection of all elements contained in the stash. For practical implementation on standard hardware, we suggest to implement the stash as a common hash table.

Tables 8, 8, and 8 depict the required stash sizes for the standard resp. simplified algorithm. Hereby, the latter algorithm is implemented in both variants, i.e. with and without self-loops. As expected, we observe that the implementation without self-loops offers better performance, but the difference is small. Furthermore, the standard data structure exhibits an even better behavior, however there is again no wide

Tab. 1: Stash sizes required for *standard cuckoo hashing* possessing two tables of size m and holding αm keys. The results are measured over a sample size of 10^7.

	stash:	0	1	2	3	4	5	6	7	8	9	10	>
$m = 500$	$\alpha = 0.40$:	9922992	73458	3312	219	18	1						
	$\alpha = 0.45$:	9677359	283258	33842	4638	778	108	17					
	$\alpha = 0.47$:	9416255	481752	81919	16037	3213	653	138	24	8	1		
	$\alpha = 0.48$:	9215990	621670	125597	28221	6565	1510	314	106	20	5	0	2
$m = 5{\cdot}10^3$	$\alpha = 0.40$:	9987684	12196	119	1								
	$\alpha = 0.45$:	9900456	93712	5359	422	48	2	1					
	$\alpha = 0.47$:	9711209	253414	29837	4563	780	159	29	7	2			
	$\alpha = 0.48$:	9488603	421374	70795	14759	3350	813	237	48	15	3	3	
$m = 5{\cdot}10^4$	$\alpha = 0.40$:	9998713	1286	1									
	$\alpha = 0.45$:	9985645	14197	156	2								
	$\alpha = 0.47$:	9936045	61276	2507	159	12	1						
	$\alpha = 0.48$:	9832037	153484	12716	1491	220	44	6	2				

Tab. 2: Stash sizes required for *simplified cuckoo hashing* possessing a tables of size $2m$ and holding αm keys. The data structure is implemented such that both storage locations of a distinct key may be equal (i.e. *with self-loops*). The results are measured over a sample size of 10^7.

	stash:	0	1	2	3	4	5	6	7	8	9	10	>
$m = 500$	$\alpha = 0.40$:	9909243	86462	4001	271	20	3						
	$\alpha = 0.45$:	9642907	313309	37468	5407	753	132	24					
	$\alpha = 0.47$:	9366101	522531	89481	17484	3482	740	156	22	1	1	0	1
	$\alpha = 0.48$:	9156407	668026	135375	30822	7073	1773	386	110	23	4	1	
$m = 5{\cdot}10^3$	$\alpha = 0.40$:	9985818	14043	138	1								
	$\alpha = 0.45$:	9892585	101327	5578	454	47	8	1					
	$\alpha = 0.47$:	9696073	266735	31215	4898	869	163	35	10	1	1		
	$\alpha = 0.48$:	9466796	439806	73228	15373	3604	891	209	63	24	5	1	
$m = 5{\cdot}10^4$	$\alpha = 0.40$:	9998397	1602	1									
	$\alpha = 0.45$:	9984851	14965	182	2								
	$\alpha = 0.47$:	9933383	63861	2573	176	7							
	$\alpha = 0.48$:	9826754	158141	13246	1601	221	25	10	1	1			

Tab. 3: Stash sizes required for *simplified cuckoo hashing* possessing a tables of size $2m$ and holding αm keys. The data structure is implemented such that both storage locations of a key are surely different (i.e. *without self-loops*). The results are measured over a sample size of 10^7.

	stash:	0	1	2	3	4	5	6	7	8	9	10	$>$
$m = 500$	$\alpha = 0.40$:	9909928	85673	4124	255	19	1						
	$\alpha = 0.45$:	9641228	314948	37399	5408	851	133	27	4	0	2		
	$\alpha = 0.47$:	9365150	523756	89367	17283	3550	702	149	33	5	4	1	
	$\alpha = 0.48$:	9154452	670044	135219	30953	7184	1655	377	88	20	7	1	
$m = 5{\cdot}10^3$	$\alpha = 0.40$:	9986083	13770	144	3								
	$\alpha = 0.45$:	9893396	100366	5675	501	53	8	1					
	$\alpha = 0.47$:	9695785	266723	31588	4847	846	174	31	6				
	$\alpha = 0.48$:	9467420	439340	73211	15388	3523	838	210	55	10	4	1	
$m = 5{\cdot}10^4$	$\alpha = 0.40$:	9998537	1461	2									
	$\alpha = 0.45$:	9984860	14972	167	1								
	$\alpha = 0.47$:	9933804	63485	2511	186	12	2						
	$\alpha = 0.48$:	9827037	157866	13249	1599	206	34	9					

difference. From the data given in all three tables, we additionally observe the following properties: For fixed $s > 0$ and m, the percentage of hash tables that demand a stash of size s increases as the load factor α increases. On the other hand, increasing m while holding α constant, decreases the expected number of items in the stash. Note that these numerical results are in good accordance with the theoretical analysis. We conclude that an additional memory of small size is sufficient to reduce the failure rate of cuckoo hashing drastically. Though, small table sizes and/or load factors close to the limit load of 0.5 demand special attention.

9 Summary and Conclusion

We showed that it is possible to use a stash without a complicated cycle-detection mechanism to determine an admissible key that can be placed in the stash. This is because of the fact that we have proved that it is sufficient to put the last kicked-out key in the stash, whenever an otherwise unresolvable situation occurs. The new insertion procedure thus offers better performance than the modified version required in Kirsch et al. (2008). As a further advantage, it is possible to break up large clusters and hence speed up further insertion procedures. Further, we adopted the analysis to simplified cuckoo hashing, a variant of the original algorithm that grants both hash functions access to the whole table. Finally, we presented a method to obtain exact asymptotic expansions of the failure rate. All these results extend the original analysis given in Kirsch et al. (2008) and verify that this algorithm offers very interesting properties.

As future work, we suggest a detailed analysis of the partial differential recursion of the generating functions of complex bipartite graphs with positive excess. Using these results, it will be possible to study the structure of these graphs.

Acknowledgements

The author would like to thank Michael Drmota for helpful suggestions on this paper. Further, I am grateful for the valueable comments received during the review process.

References

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, Mass., London, England, second edition, 2001.

L. Devroye and P. Morin. Cuckoo hashing: Further analysis. *Information Processing Letters*, 86(4): 215–219, 2003.

M. Dietzfelbinger and U. Schellbach. On risks of using cuckoo hashing with simple universal hash classes. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 795–804. SIAM, 2009.

M. Dietzfelbinger and C. Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007.

M. Drmota. A bivariate asymptotic expansion of coefficients of powers of generating functions. *European Journal of Combinatorics*, 15(2):139–152, 1994.

M. Drmota and R. Kutzelnigg. A precise analysis of cuckoo hashing. *ACM Transactions on Algorithms*, 2009. submitted.

P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, Cambridge, UK, 2009.

D. Fotakis, R. Pagh, P. Sanders, and P. G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005.

D. Gardy. Some results on the asymptotic behaviour of coefficients of large powers of functions. *Discrete Mathematics*, 139(1-3):189–217, 1995.

O. Gimenez, A. de Mier, and M. Noy. On the number of bases of bicircular matroids. *Annals of Combinatorics*, 9(1):35–45, 2005.

I. J. Good. Saddle-point methods for the multinomial distribution. *Ann. Math. Stat.*, 28(4):861–881, 1957.

S. Janson, D. E. Knuth, T. Łuczak, and B. Pittel. The birth of the giant component. *Random Structures and Algorithms*, 4(3):233–359, 1993.

A. Kirsch, M. Mitzenmacher, and U. Wieder. More robust hashing: Cuckoo hashing with a stash. In *Proceedings of the 16th Annual European Symposium on Algorithms*, 2008.

D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, Boston, second edition, 1998.

R. Kutzelnigg. *Random Graphs and Cuckoo Hashing*. Südwestdeutscher Verlag für Hochschulschriften, Saarbrücken, 2009. ISBN 978-3-8381-0207-8.

R. Kutzelnigg. An improved version of cuckoo hashing: Average case analysis of construction cost and search operations. *Mathematics in Computer Science*, 3(1):47–60, 2010.

R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

A Asymptotic Expansions via the Saddle Point Method

This appendix provides results that are required to infer asymptotic expansions of the coefficients of generating functions counting the number of cuckoo graphs without "bad" components. These results can be obtained using a saddle point approach, see, e.g., Drmota (1994); Flajolet and Sedgewick (2009); Gardy (1995); Good (1957) for details concerning this method and Kutzelnigg (2009) for proofs of the lemmata at full length. Note that further coefficients of the asymptotic expansions can be calculated in the same way, but the expressions are so complicated that it does not make sense to provide them outside a computer algebra system. A maple worksheet is available on request from the author.

Lemma 13 *Let $f(x)$ and $g(x)$ be analytic functions locally around 0 such that all coefficients $[x^m]f(x)$ and $[x^m]g(x)$ are non negative, $f(0) \neq 0$, and such that the "aperiodicity condition" $\gcd\{m|[x^m]f(x) > 0\} = 1$ holds.*

Let R be a compact interval of the positive real line that is contained in the radius of convergence of $f(x)$ and $g(x)$. Furthermore set

$$S = \left\{ \frac{x}{f(x)} \frac{\partial}{\partial x} f(x) : x \in R \right\}.$$

Then we have

$$[x^m]g(x)f(x)^k = \frac{g(x_0)f(x_0)^k}{x_0^m \sqrt{2\pi k \kappa_2}} \left(1 + \frac{H}{24\kappa_2^3}\frac{1}{k} + \mathcal{O}\left(\frac{1}{k^2}\right)\right),$$

uniformly for $m/k \in S$, where x_0 is uniquely determined by

$$\frac{m}{k} = \frac{x_0 f'(x_0)}{f(x_0)}.$$

and the constants κ_2 and H are given in the following way. Let κ_i and $\overline{\kappa}_i$ be the cummulants

$$\kappa_i = \left[\frac{\partial^i}{\partial u^i} \log f(x_0 e^u)\right]_{u=0}, \qquad \overline{\kappa}_i = \left[\frac{\partial^i}{\partial u^i} \log g(x_0 e^u)\right]_{u=0}.$$

Then H is given by

$$12\kappa_2\kappa_3\overline{\kappa}_1 + 3\kappa_2\kappa_4 - 12\kappa_2^2\overline{\kappa}_1^2 - 12\kappa_2^2\overline{\kappa}_2 - 5\kappa_3^2.$$

Lemma 14 *Let $f(x,y)$ and $g(x,y)$ be analytic functions locally around $(x,y) = (0,0)$ such that all coefficients $[x^{m_1}y^{m_2}]f(x,y)$ and $[x^{m_1}y^{m_2}]g(x,y)$ are non negative and that there exists M such that all indices (m_1, m_2) with $m_1, m_2 \geq M$ can be represented as a finite linear combination of the set $\{(m_1, m_2)|[x^{m_1}y^{m_2}]f(x,y) > 0\}$ with positive integers as coefficients.*

Let R_1 and R_2 be compact intervals of the positive real line such that $R = R_1 \times R_2$ is contained in the regions of convergence of $f(x,y)$ and $g(x,y)$. Furthermore set

$$S = \left\{ \left(\frac{x}{f(x,y)} \frac{\partial}{\partial x} f(x,y), \frac{y}{f(x,y)} \frac{\partial}{\partial y} f(x,y) \right) : (x,y) \in R \right\}.$$

Then we have

$$[x^{m_1}y^{m_2}]g(x,y)f(x,y)^k = \frac{g(x_0,y_0)f(x_0,y_0)^k}{2\pi x_0^{m_1} y_0^{m_2} k\sqrt{\Delta}} \left(1 + \frac{H}{24\Delta^3}\frac{1}{k} + \mathcal{O}\left(\frac{1}{k^2}\right)\right),$$

uniformly for $(m_1/k, m_2/k) \in S$*, where* x_0 *and* y_0 *are uniquely determined by*

$$\frac{m_1}{k} = \frac{x_0}{f(x_0,y_0)} \left[\frac{\partial}{\partial x} f(x,y)\right]_{(x_0,y_0)}, \qquad \frac{m_2}{k} = \frac{y_0}{f(x_0,y_0)} \left[\frac{\partial}{\partial y} f(x,y)\right]_{(x_0,y_0)}$$

and the constants Δ *and* H *are given in the following way: Let* κ_{ij} *and* $\overline{\kappa}_{ij}$ *be the cummulants*

$$\kappa_{ij} = \left[\frac{\partial^i}{\partial u^i}\frac{\partial^j}{\partial v^j} \log f(x_0 e^u, y_0 e^v)\right]_{(0,0)}, \qquad \overline{\kappa}_{ij} = \left[\frac{\partial^i}{\partial u^i}\frac{\partial^j}{\partial v^j} \log g(x_0 e^u, y_0 e^v)\right]_{(0,0)}.$$

Then $\Delta = \kappa_{20}\kappa_{02} - \kappa_{11}^2$ *holds and* H *is given by*

$$H = \alpha + \beta + \hat{\beta} + \gamma\overline{\kappa}_{10} + \hat{\gamma}\overline{\kappa}_{01} + \delta\overline{\kappa}_{10}\overline{\kappa}_{01} + \eta\overline{\kappa}_{10}^2 + \hat{\eta}\overline{\kappa}_{01}^2 + 4\eta\overline{\kappa}_{20} + 4\hat{\eta}\overline{\kappa}_{02} + 4\delta\overline{\kappa}_{11},$$

where

$$\begin{aligned}
\alpha &= 54\kappa_{21}\kappa_{11}\kappa_{12}\kappa_{20}\kappa_{02} + 6\kappa_{22}\kappa_{20}\kappa_{02}\kappa_{11}^2 - 12\kappa_{22}\kappa_{11}^4 + 4\kappa_{03}\kappa_{11}^3\kappa_{30} \\
&\quad + 36\kappa_{21}\kappa_{11}^3\kappa_{12} + 6\kappa_{22}\kappa_{20}^2\kappa_{02}^2 + 6\kappa_{03}\kappa_{11}\kappa_{30}\kappa_{20}\kappa_{02}, \\
\beta &= -5\kappa_{02}^3\kappa_{30}^2 + 30\kappa_{02}^2\kappa_{30}\kappa_{11}\kappa_{21} - 24\kappa_{02}\kappa_{30}\kappa_{12}\kappa_{11}^2 - 6\kappa_{02}^2\kappa_{30}\kappa_{12}\kappa_{20} \\
&\quad - 12\kappa_{11}\kappa_{02}^2\kappa_{31}\kappa_{20} - 36\kappa_{02}\kappa_{21}^2\kappa_{11}^2 - 9\kappa_{02}^2\kappa_{21}^2\kappa_{20} + 3\kappa_{02}^3\kappa_{40}\kappa_{20} \\
&\quad - 3\kappa_{02}^2\kappa_{40}\kappa_{11}^2 + 12\kappa_{11}^3\kappa_{02}\kappa_{31}, \\
\gamma &= 12\Delta\left(\kappa_{02}^2\kappa_{30} - \kappa_{11}\kappa_{20}\kappa_{03} - 3\kappa_{21}\kappa_{11}\kappa_{02} + \kappa_{12}\kappa_{11}^2 + \kappa_{12}(\kappa_{02}\kappa_{20} + \kappa_{11}^2)\right), \\
\delta &= 24\Delta(\kappa_{11}\kappa_{20}\kappa_{02} - \kappa_{11}^3), \\
\eta &= 12\Delta(\kappa_{02}\kappa_{11}^2 - \kappa_{02}^2\kappa_{20}),
\end{aligned}$$

and $\hat{}$ *indicates to replace all functions of type* κ_{ij} *by* κ_{ji}*.*