An Analysis of Random-Walk Cuckoo Hashing

Alan Frieze[1]*, Páll Melsted[1], and Michael Mitzenmacher[2]**

[1] Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh PA 15213
U.S.A.
[2] School of Engineering and Applied Sciences
Harvard University
Cambridge MA 02138

Abstract. In this paper, we provide a polylogarithmic bound that holds with high probability on the insertion time for cuckoo hashing under the random-walk insertion method. Cuckoo hashing provides a useful methodology for building practical, high-performance hash tables. The essential idea of cuckoo hashing is to combine the power of schemes that allow multiple hash locations for an item with the power to dynamically change the location of an item among its possible locations. Previous work on the case where the number of choices is larger than two has required a breadth-first search analysis, which is both inefficient in practice and currently has only a polynomial high probability upper bound on the insertion time. Here we significantly advance the state of the art by proving a polylogarithmic bound on the more efficient random-walk method, where items repeatedly kick out random blocking items until a free location for an item is found.
Key Words: Cuckoo Hashing, Random Walk Algorithm.

1 Introduction

Cuckoo hashing [12] provides a useful methodology for building practical, high-performance hash tables by combining the power of schemes that allow multiple hash locations for an item (e.g., [1–3, 13]) with the power to dynamically change the location of an item among its possible locations. Briefly (more detail is given in Section 2), each of n items x has d possible locations $h_1(x), h_2(x), \ldots, h_d(x)$, where d is typically a small constant and the h_i are hash functions, typically assumed to behave as independent fully random hash functions. (See [11] for some justification of this assumption.) We assume each location can hold only one item. When an item x is inserted into the table, it can be placed immediately if one of its d locations is currently empty. If not, one of the items in its d locations must be displaced and moved to another of its d choices to make room for x. This item in turn may need to displace another item out of one its d locations. Inserting an item may require a sequence of moves, each maintaining the invariant that each item remains in one of its d potential locations, until no further evictions are needed. Further variations of cuckoo hashing, including possible implementation designs, are considered in for example [5], [6], [7], [8], [9].

It is often helpful to place cuckoo hashing in a graph theoretic setting, with each item corresponding to a node on one side of a bipartite graph, each bucket corresponding to a node on the other side of a bipartite graph, and an edge between an item x and a bucket b if b is one of the d buckets where x can be placed. In this case, an assignment of items to buckets forms a matching and a sequence of moves that allows a new item to be placed corresponds to a type of augmenting path in this graph. We call this the cuckoo graph (and define it more formally in Section 2).

The case of $d = 2$ choices is notably different than for other values of d. When $d = 2$, after the first choice of an item to kick out has been made, there are no further choices as one walks through the cuckoo graph to find an augmenting path. Alternatively, in this case one can think of the cuckoo graph in an alternative form, where the only nodes are buckets and items correspond to edges between the buckets, each item connecting the two buckets corresponding to it. Because of these special features of the $d = 2$ case, its analysis appears much simpler, and the theory for the case where there are $d = 2$ bucket choices for each item is well understood at this point [4, 10, 12].

* Supported in part by NSF Grant CCF-0502793.
** Supported in part by NSF Grant CNS-0721491 and grants from Cisco Systems Inc., Yahoo!, and Google.

The case where $d > 2$ remains less well understood, although values of d larger than 2 rate to be important for practical applications. The key question is if when inserting a new item x all $d > 2$ buckets for x are already full, what should one do? A natural approach in practice is to pick one of the d buckets randomly, replace the item y at that bucket with x, and then try to place y in one of its other $d-1$ bucket choices [6]. If all of the buckets for y are full, choose one of the other $d-1$ buckets (other than the one that now contains x, to avoid the obvious cycle) randomly, replace the item there with y, and continue in the same fashion. At each step (after the first), place the item if possible, and if not randomly exchange the item with one of $d-1$ choices. We refer to this as the random-walk insertion method for cuckoo hashing.

There is a clear intuition for how this random walk on the buckets should perform. If a fraction f of the items are adjacent to at least one empty bucket in the corresponding graph, then we might expect that each time we place one item and consider another, we should have approximately a probability f of choosing an item adjacent to an empty bucket. With this intuition, assuming the load of the hash table is some constant less than 1, the time to place an item would be at most $O(\log n)$ with high probability [3].

Unfortunately, it is not clear that this intuition should hold true; the intuition assumes independence among steps when the assumption is not necessarily warranted. Bad substructures might arise where a walk could be trapped for a large number of steps before an empty bucket is found. Indeed, analyzing the random-walk approach has remained open, and is arguably the most significant open question for cuckoo hashing today.

Because the random-walk approach has escaped analysis, thus far the best analysis for the case of $d > 2$ is due to Fotakis et al. [6], and their algorithm uses a breadth-first search approach. Essentially, if the d choices for the initial item x are filled, one considers the other choices of the d items in those buckets, and if all those buckets are filled, one considers the other choices of the items in those buckets, and so on. They prove a constant expected time bound for an insertion for a suitably sized table and constant number of choices, but to obtain a high probability bound under their analysis requires potentially expanding a logarithmic number of levels in the breadth-first search, yielding only a polynomial bound on the time to find an empty bucket with high probability. It was believed this should be avoidable by analyzing the random-walk insertion method. Further, in practice, the breadth-first search would not be the choice for most implementations because of its increased complexity and memory needs over the random-walk approach.

In this paper, we demonstrate that, with high probability, for sufficiently large d the cuckoo graph has certain structural properties that yield that on the insertion of any item, the time required by the random-walk insertion method is polylogarithmic in n. The required properties and the intuition behind them are given in subsequent sections. Besides providing an analysis for the random-walk insertion method, our result can be seen as an improvement over [6] in that the bound holds for every possible starting point for the insertion (with high probability). The breadth-first search of [6] gives constant expected time, implying polylogarithmic time with probability $1 - o(1)$. However when inserting $\Omega(n)$ element into the hash table, the breadth-first search algorithm cannot guarantee a sub-polynomial running time for the insertion of each element. This renders the breadth-first search algorithm unsuitable for many applications that rely on guarantees for individual insertions and not just expected or amortized time complexities.

While the results of [6] provide a starting point for our work, we require further deconstruction of the cuckoo graph to obtain our bound on the performance of the random-walk approach.

Simulations in [6] (using the random-walk insertion scheme), indicate that constant expected insertion time is possible. While our guarantees do not match the running time observed in simulations, they give the first clear step forward on this problem for some time.

2 Definitions and Results

We begin with the relevant definitions, followed by a statement of and explanation of our main result.

Let $h_1, \ldots h_d$ be independent fully random hash functions $h_i : [n] \to [m]$ where $m = (1+\varepsilon)n$. The necessary number of choices d will depend on ϵ, which gives the amount of extra space in the table. We let the *cuckoo graph* G be a bipartite graph with a vertex set $L \cup R$ and an edge set $\bigcup_{x \in L}\{(x, h_1(x)), \ldots (x, h_d(x)\}$,

[3] An event $\mathcal{E}_n$ occurs *with high probability* if $\mathsf{P}(\mathcal{E}_n) = 1 - O(1/n^\alpha)$ for some constant $\alpha > 0$, see also discussion on page 3

where $L = [n]$ and $R = [m]$. We refer to the left set L of the bipartite graph as *items* and the right set R as *buckets*.

An assignment of the items to the buckets is a left-perfect matching M of G such that every item $x \in L$ is incident to a matching edge. The vertices $F \subseteq R$ not incident to the matching M are called *free* vertices. For a vertex v the *distance* to a free vertex is the shortest M-alternating path from v to a free vertex.

We present the algorithm for insertion as Algorithm 1 below. The algorithm augments the current matching M with an augmenting path P. An item is assigned to a free neighbor if one exists; otherwise, a random neighbor is chosen to displace from its bucket, and this is repeated until an augmenting path is found. In practice, one generally sets an upper bound on the number of moves allowed to the algorithm, and a failure occurs if there remains an unassigned item after that number of moves. Such failure can be handled by additional means, such as stashes [8].

Algorithm 1 Insert-node

```
1: procedure INSERT-NODE(G,M,u)
2:     P ← ()
3:     v ← u
4:     i ← d + 1
5:     loop
6:         if h_j(v) is not covered by M for some j ∈ {1,...,d} then
7:             P ← P ⊕ (v, h_j(v))
8:             return Augment(M,P)
9:         else
10:            Let j ∈_R {1,...,d} \ {i} and w be such that (h_j(v), w) ∈ M
11:            P ← P ⊕ (v, h_j(v)) ⊕ (h_j(v), w)
12:            v ← w
13:            i ← j
14:        end if
15:    end loop
16: end procedure
```

We note that our analysis that follows also holds when the table experiences deletions. This is because our result is based on the structure of the underlying graph G, and not on the history that led to the specific current matching. The statement of the main result is that given that G satisfies certain conditions, which it will with high probability, the insertion time is polylogarithmic with high probability. It is important to note that we have two distinct probability spaces, one for the hash functions which induce the graph G, and another for the randomness employed by the algorithm. For the probability space of hash function, we say that an event $\mathcal{E}_n$ occurs *with high probability* if $\mathsf{P}(\mathcal{E}_n) = 1 - O(n^{-2d})$. For the probability space of randomness used by the algorithm we use the regular definition of *with high probability*.

Theorem 1. *Conditioned on an event of probability* $1 - O(n^{4-2d})$ *regarding the structure of the cuckoo graph* G*, the expected time for insertion into a cuckoo hash-table using Algorithm 1 is* $O\left(\log^{1+\gamma_0+2\gamma_1} n\right)$ *where* $\gamma_0 = \frac{d+\log d}{(d-1)\log(d/3)}$ *and* $\gamma_1 = \frac{d+\log d}{(d-1)\log(d-1)}$*, assuming* $d \geq 8$ *and if* $\varepsilon \leq \frac{1}{6}$*,* $d \geq 4 + 2\varepsilon - 2(1+\varepsilon)\log\left(\frac{\varepsilon}{1+\varepsilon}\right)$*. Furthermore, the insertion time is* $O\left(\log^{2+\gamma_0+2\gamma_1} n\right)$ *with high probability.*

The algorithm will fail if the graph does not have a left-perfect matching, which happens with probability $O(n^{4-2d})$ [6]. We show that all necessary structural properties of G hold with probability $1 - O(n^{-2d})$, so that the probability of failure is dominated by the probability that G has no left-perfect matching.

At a high level, our argument breaks down into a series of steps. First, we show that the cuckoo graph expands suitably so that most vertices are within $O(\log \log n)$ distance from a free vertex. Calling the free vertices F and this set of vertices near to the free vertices S, we note that if reach a vertex in S, then the probability of reaching F from there over the next $O(\log \log n)$ steps in the random walk process is inverse polylogarithmic in n, so we have a reasonable chance of getting to a free vertex and finishing. We next show that if the cuckoo graph has an expansion property, then from any starting vertex, we are likely to reach

a vertex of S though the random walk in only $O(\log n)$ steps. This second part is the key insight into this result; instead of trying to follow the intuition to reach a free vertex in $O(\log n)$ steps, we aim for the simpler goal of reaching a vertex S close to F and then complete the argument.

As a byproduct of our Lemma 2 (below), we get an improved bound on the expected running time for the breadth-first variation on cuckoo hashing from [6], $\left(\frac{1}{\varepsilon}\right)^{O(1)}$ instead of $\left(\frac{1}{\varepsilon}\right)^{O(\log d)}$.

Theorem 2. *The breadth-first search insertion procedure given in [6] runs in* $O\left(\max\left\{d^4\left(\frac{1}{6\varepsilon}\right)^{\frac{1}{1-\frac{\log 6}{\log d}}}, d^5\right\}\right)$ *expected time, provided* $d \geq 8$ *and if* $\varepsilon \leq \frac{1}{6}$ *if* $d \geq 4 + 2\varepsilon - 2(1+\varepsilon)\log\left(\frac{\varepsilon}{1+\varepsilon}\right)$.

We prove the necessary lemmas below. The first lemma shows that large subsets of R have large neighborhoods in L. Using this we can show in our second lemma that the number of R-vertices at distance k from F shrinks geometrically with k. This shows that the number of vertices at distance $\Omega(\log\log n)$ from F is sufficiently small. The second lemma shows that at least half the vertices of L are at a constant distance from F. The next two lemmas will be used to show that successive levels in a breath first search expand very fast i.e. at a rate close to $d-1$. The first of these lemmas deals with the first few levels of the process by showing that small connected subgraphs cannot have to many "extra" edges. The second will account for subsequent levels through expansion.

3 Expansion and Related Graph Structure

We first show that large subsets of R have corresponding large neighborhoods in L.

Lemma 1. *If* $1/2 \leq \beta \leq 1 - \frac{2d^3\log n}{n}$ *and* $\alpha = d - 1 - \frac{d+\log d}{1-\log(1-\beta)} > 0$ *then with high probability every subset* $Y \subseteq R$ *of size* $|Y| = (\beta+\varepsilon)n$, *has a* G*-neighborhood* $X \subseteq L$ *of size at least* $n\left(1 - \frac{1-\beta}{\alpha}\right)$.

Proof. We show by a union bound that with high probability, there does not exist a pair X, Y such that $|Y| = (\beta+\varepsilon)n$, $|X| < n - \frac{(1+\varepsilon)n-|Y|}{\alpha}$ and X is the neighborhood of Y in G.

Let $S = L \setminus X$ and $T = R \setminus Y$. Then $|S| \geq \frac{(1+\varepsilon)n-|Y|}{\alpha} = \frac{n(1-\beta)}{\alpha}$ and $|T| = (1+\varepsilon)n - (\beta+\varepsilon)n = (1-\beta)n$. Each vertex in L has all of its edges in T with probability $\left(\frac{1-\beta}{1+\varepsilon}\right)^d$ independently of other vertices. Thus for any T the size of S is a binomially distributed random variable, $\mathrm{Bin}(n, \left(\frac{1-\beta}{1+\varepsilon}\right)^d)$. Thus the probability of the existence of a pair X, Y is at most

$$\begin{aligned}\binom{(1+\varepsilon)n}{(\beta+\varepsilon)n}\mathsf{P}\left(|S| \geq \frac{(1-\beta)n}{\alpha}\right) &= \binom{(1+\varepsilon)n}{(1-\beta)n}\mathsf{P}\left(\mathrm{Bin}\left(n, \left(\frac{1-\beta}{1+\varepsilon}\right)^d\right) \geq \frac{(1-\beta)n}{\alpha}\right) \\ &\leq \left(e\frac{1+\varepsilon}{1-\beta}\right)^{(1-\beta)n}\left(e\frac{\left(\frac{1-\beta}{1+\varepsilon}\right)^d}{\frac{1-\beta}{\alpha}}\right)^{\frac{1-\beta}{\alpha}n} \\ &= \left(\frac{\alpha e^{1+\alpha}(1+\varepsilon)^{\alpha-d}}{(1-\beta)^{\alpha-d+1}}\right)^{\frac{1-\beta}{\alpha}n} \qquad (1)\end{aligned}$$

where we have used the inequality $\mathsf{P}\left(\mathrm{Bin}(n,p) \geq \rho pn\right) \leq \left(\frac{e}{\rho}\right)^{\rho pn}$. (While there are tighter bounds, this is sufficient for our purposes.)

Taking logarithms, dropping the $1+\varepsilon$ factor, and letting $D = d + \log d$ and $L = \log(1-\beta)$ gives

$$\begin{aligned}\frac{\log((1))}{\frac{1-\beta}{\alpha}n} &\leq \log\left(\frac{d-1-D/(1-L)}{d}\right) + D - \frac{D}{1-L} + \frac{DL}{1-L} \\ &= \log\left(\frac{d-1-D/(1-L)}{d}\right) \leq \log\left(\frac{d-1}{d}\right)\end{aligned}$$

Then we can upper bound the expression in (1) by

$$\left(\frac{d-1}{d}\right)^{\frac{1-\beta}{\alpha}n} \leq \exp\left(-\frac{1}{d}\frac{2d^3\log n}{d}n\right) \leq n^{-2d}$$

The following Lemma corresponds to Lemma 8 in [6]. We give an improved bound on an important parameter k^* which gives an improvement for the running time of the breadth-first search algorithm.

Lemma 2. *Assume $d \geq 8$ and furthermore if $\varepsilon \leq \frac{1}{6}$ we assume $d \geq 4 + 2\varepsilon - 2(1+\varepsilon)\log\left(\frac{\varepsilon}{1+\varepsilon}\right)$. Then the number of vertices in L at distance at most k^* from F is at least $\frac{n}{2}$, where $k^* = 4 + \frac{\log\left(\frac{1}{6\varepsilon}\right)}{\log\left(\frac{d}{6}\right)}$ if $\varepsilon \leq \frac{1}{6}$ and $k^* = 5$ if $\varepsilon \geq \frac{1}{6}$.*

Proof

Let M be any left-perfect matching of the cuckoo graph G. Let Y_0 be the free vertices in R and $X_1 = N(Y_0)$. The vertices of X_1 are adjacent to free vertices, and thus are at augmentation distance 1. Let Y_1 be the matching neighbors of vertices in X_1 in addition to Y_0. In general given X_i we let

$$Y_i = Y_0 \cup \{y \in R : (x,y) \in M \text{ for some } x \in X_i\} \qquad \text{and} \qquad X_{i+1} = N(Y_i)$$

Note that $|Y_0| = \varepsilon$ and $|X_i| = |Y_i| + \varepsilon n$ for $i \geq 1$. Thus to show that $|X_i| \geq \frac{n}{2}$ for some i it is enough to concentrate on expansion properties of the Y sets in R.

Claim. Any set $Y \subseteq R$ of size $x(1+\varepsilon)n$ has a neighborhood of size at least

$$\begin{cases} \frac{d}{6}xn & \text{if } x \leq \frac{6}{d^2} & \textbf{Case 1} \\ \frac{1}{d}n & \text{if } \frac{6}{d^2} \leq x \leq \frac{1}{d} & \textbf{Case 2} \\ \frac{1}{5}n & \text{if } \frac{1}{d} \leq x \leq \frac{1}{5} & \textbf{Case 3} \\ \frac{3}{10}n & \text{if } \frac{1}{5} \leq x \leq \frac{3}{10} & \textbf{Case 4} \\ \frac{2}{5}n & \text{if } \frac{3}{10} \leq x \leq \frac{2}{5} & \textbf{Case 5} \\ \frac{1}{2}n & \text{if } \frac{2}{5} \leq x & \textbf{Case 6} \end{cases}$$

Using Claim 3 we see that it will take at most $\log_{\frac{d}{6}}\left(\frac{\frac{6}{d^2}}{\varepsilon}\right)$ steps to go from size εn to at least $\frac{6}{d^2}$ and 5 additional steps to go from $\frac{6}{d^2}n$ to $\frac{1}{2}n$. This makes for a total of at most

$$\begin{aligned} i^* &\leq \log_{\frac{d}{6}}\left(\frac{\frac{6}{d^2}}{\varepsilon}\right) + 6 \\ &= \frac{-\log(\varepsilon) - 2\log d + \log 6}{\log\left(\frac{d}{6}\right)} + 6 \\ &= 4 + \frac{\log\left(\frac{1}{6\varepsilon}\right)}{\log\left(\frac{d}{6}\right)} \end{aligned}$$

steps to go from Y_0 of size εn to X_{i^*} of size at least $\frac{1}{2}n$. Note that if $\varepsilon \geq \frac{1}{6}$ we can use $i^* \leq 5$, since then $x \geq \frac{1}{7} \geq \frac{1}{d}$.

Proof of claim 3

Assume $d \geq 8$ and if $\varepsilon \leq \frac{1}{6}$ we require $d \geq 4 + 2\varepsilon - 2(1+\varepsilon)\log\left(\frac{\varepsilon}{1+\varepsilon}\right)$. For $\varepsilon \leq 1$ this implies $\log\left(\frac{\varepsilon}{1+\varepsilon}\right) \geq 1 - \frac{d-2}{2(1+\varepsilon)}$. Now let $Y \subset R$, $|Y| = x(1+\varepsilon)n$ and assume $|Y| \geq \varepsilon n$. So $\frac{\varepsilon}{1+\varepsilon} \leq x$ and for $\varepsilon \geq 1$ we have $x \geq \frac{1}{2}$.

We upper bound the probability than any set Y, has than sn neighbours in L by

$$\begin{aligned} &\binom{(1+\varepsilon)n}{x(1+\varepsilon)n}\binom{n}{sn}\left((1-x)^d\right)^{(1-s)n}\left(1-(1-x)^d\right)^{sn} \qquad (2) \\ &\leq \exp\left(n\left((1+\varepsilon)H(x) + H(s) + (1-s)d\log(1-x) + s\log(1-(1-x)^d)\right)\right) \end{aligned}$$

where $H(x) = -x\log(x) - (1-x)\log(1-x)$ and $\binom{n}{k} \le e^{nH\left(\frac{k}{n}\right)}$. We will assume that

$$s \le 1 - (1-x)^d. \tag{3}$$

The product of the terms involving s in (2) is monotone increasing up to this point.

It is enough to show that the factor

$$\Phi_1 = (1+\varepsilon)H(x) + H(s) + (1-s)d\log(1-x) + s\log(1-(1-x)^d) \tag{4}$$

is strictly negative.

It is enough to show that Φ_1 is strictly negative for some particular value of $s^* \le 1-(1-x)^d$. Inflating by a factor n for smaller s we get a bound $O(ne^{-\Omega(n)}) = e^{-\Omega(n)}$.

Case 1: $\frac{\varepsilon}{1+\varepsilon} \le x \le \frac{6}{d^2}$.

We start with (4) and write $s = \gamma dx$ and assume $s \le \frac{1}{d}$. We also use the upper bound $H(x) \le x(1-\log x)$ and get

$$(1+\varepsilon)x(1-\log x) + \gamma dx(1-\log(\gamma dx)) + (1-\gamma dx)d\log(1-x) + \gamma dx\log(1-(1-x)^d) \tag{5}$$

Now use $\log(1-x) \le -x - \frac{x^2}{2}$, $(1-\gamma dx)d \ge (1-\frac{1}{d})d = d-1$ and $\log(1-(1-x)^d) \le \log dx$ to obtain

$$\Phi_1 \le \Phi_2 = x\left((1+\varepsilon)(1-\log x) + \gamma d(1-\log\gamma) - (d-1)\right) - (d-1)\frac{x^2}{2} \tag{6}$$

Since $x \ge \frac{\varepsilon}{1+\varepsilon}$ and $(1+\varepsilon)\left(1-\log\left(\frac{\varepsilon}{1+\varepsilon}\right)\right) \le \frac{d-2}{2}$, we see that using $\gamma = 6$ gives $\gamma(1-\log\gamma) < 1/2$ and so the coefficient of x in (6) is negative.

Case 2: $\frac{6}{d^2} \le x \le \frac{1}{d}$.

If $|Y| \in \left[\frac{6n}{d^2}, \frac{n}{d}\right]$ we can choose a subset Y' of Y of size exactly $\frac{6n}{d^2}$ and use $|N(Y)| \ge |N(Y')| \ge \frac{n}{d}$.

Case 3: $\frac{1}{d} \le x$ and $s \le 1/5$.

Φ_1 is increasing with respect to ε and $x \ge \frac{\varepsilon}{1+\varepsilon}$. So we can take $\varepsilon = \frac{x}{1-x}$ in order to bound Φ_1.

$$\Phi_1 \le \Phi_3 = \frac{1}{1-x}H(x) + H(s) + (1-s)d\log(1-x) + s\log(1-(1-x)^d) \tag{7}$$

The derivative of the RHS of (7) with respect to x is given by

$$d\left(\frac{s(1-x)^{d-1}}{1-(1-x)^d} - \frac{1-s}{1-x}\right) - \frac{\log x}{(1-x)^2} = \frac{-d(1-x)\left(1-\frac{s}{1-(1-x)^d}\right) - \log x}{(1-x)^2} \tag{8}$$

Note that both $-d(1-\frac{s}{1-(1-x)^d})$ and $-\frac{\log x}{1-x}$ are decreasing in x, so it is enough to verify that the numerator in the RHS of (8) is negative at the left endpoint $x = 1/d$.

For $s \le \frac{1}{5}$ we get

$$\begin{aligned} &-d\left(1-\frac{1}{d}\right)\left(1-\frac{s}{1-(1-\frac{1}{d})^d}\right) + \log d \\ \le &-(d-1)\left(1-\frac{0.2}{1-e^{-1}}\right) + \log d \quad \le -0.68(d-1) + \log d \end{aligned}$$

which is decreasing and negative for $d \ge 8$.

So for $s \le \frac{1}{5}$ we see that Φ_3 is decreasing in x. Since $x \ge \frac{1}{d}$, the maximum is obtained at $x = \frac{1}{d}$ and plugging in $x = \frac{1}{d}$ yields.

$$\Phi_4 = \frac{d}{d-1}H\left(\frac{1}{d}\right) + H(s) + (1-s)d\log\left(1-\frac{1}{d}\right) + s\log\left(1-\left(1-\frac{1}{d}\right)^d\right) \tag{9}$$

Taking the derivative of Φ_4 with respect to d gives

$$\frac{(d-1)(1-(1-\frac{1}{d})^d - s)(1+(d-1)\log(1-\frac{1}{d})) - (\log d)(1-(1-\frac{1}{d})^d))}{(d-1)^2(1-(1-\frac{1}{d})^d)} \tag{10}$$

Note that

$$0 \leq (d-1)\left(1+(d-1)\log\left(1-\frac{1}{d}\right)\right) \leq (d-1)/d.$$

We want to show that $\Phi_4' < 0$ and so we can drop the contribution from s. Factoring $1-(1-1/d)^d$ from the remainder of the numerator gives at most $(d-1)/d - \log d < 0$.

For an upper bound we plug in $d = 8$ into equation (9) and get

$$\frac{8}{7}H\left(\frac{1}{8}\right) + H(s) + (1-s)8\log\left(\frac{7}{8}\right) + s\log\left(1-\left(\frac{7}{8}\right)^8\right) \tag{11}$$

The derivative with respect to s is

$$\log\left(\frac{1-s}{s}\right) + s\log\left(\frac{1-\left(\frac{7}{8}\right)^8}{\left(\frac{7}{8}\right)^8}\right)$$

which is positive for $s \leq 1/5$. Thus plugging in $s = \frac{1}{5}$ we get

$$\frac{8}{7}H\left(\frac{1}{8}\right) + H\left(\frac{1}{5}\right) + \frac{32}{5}\log\left(\frac{7}{8}\right) + \frac{1}{5}\log\left(1-\left(\frac{7}{8}\right)^8\right) < -0.006$$

which is strictly negative.

Case 4: $1/5 \leq x \leq 3/10$.

Starting from (7) we see that the derivative of Φ_3 with respect to d is

$$\left(1 - \frac{s}{1-(1-x)^d}\right)\log(1-x)$$

which is negative since we are assuming $s \leq 1-(1-x)^d$. So we will use $d = 8$ from now on to get an upper bound on Φ_3 and obtain

$$\Phi_3 \leq \Phi_5 = \frac{1}{1-x}H(x) + H(s) + 8(1-s)\log(1-x) + s\log(1-(1-x)^8) \tag{12}$$

The derivative of Φ_5 with respect to x is

$$\Phi_5' = \frac{-8(1-x)\left(1-\frac{s}{1-(1-x)^8}\right) - \log x}{(1-x)^2} \tag{13}$$

If $\frac{1}{5} \leq x \leq 1$ and $s \leq \frac{1}{2}$, then we can upper bound the numerator by

$$-8(1-x)\left(1-\frac{\frac{1}{2}}{1-\left(\frac{4}{5}\right)^8}\right) - \log x \leq -3(1-x) - \log x$$

which is convex and negative at both endpoints, $x = 1/5$ and $x = 1$. Thus Φ_5 is decreasing in x on $[\frac{1}{5}, 1]$. Evaluating Φ_5 with $x = \frac{1}{5}$ and $s = \frac{3}{10}$ gives $\Phi_5 < -0.06$.

Case 5: $3/10 \leq x \leq 2/5$.

We evaluate Φ_5 at $x = \frac{3}{10}$ and $s = \frac{2}{5}$, and get $\Phi_5 < -0.19$.

Case 6: $2/5 \leq x \leq 1$.

Evaluating Φ_5 with $x = \frac{2}{5}$ and $s = \frac{1}{2}$ gives $\Phi_5 < -.23$. □

We can now give a proof of Theorem 2:

Proof We follow the proof of Theorem 1 in [6]. The breadth-first search insertion procedure takes time $O(|T_v|)$ where T_v is a BFS tree rooted at the newly inserted vertex v, which is grown until a free vertex is found.

The expected size of T_v is bounded above by d^{k^*} which is at most d^5 for $\varepsilon \geq \frac{1}{6}$ and

$$d^{4+\log\left(\frac{1}{6\varepsilon}\right)/\log\left(\frac{d}{6}\right)} = d^4\left(\frac{1}{6\varepsilon}\right)^{\frac{\log d}{\log d - \log 6}} = d^4\left(\frac{1}{6\varepsilon}\right)^{\frac{1}{1-\frac{\log 6}{\log d}}}$$

if $\varepsilon \leq \frac{1}{6}$. □

Let $k^* = \max\{4 + \frac{\log\left(\frac{1}{6\varepsilon}\right)}{\log\left(\frac{d}{6}\right)}, 5\}$ and let Y_k be the vertices in R at distance at most $k^* + k$ from F and let $|Y_k| = (\beta_k + \varepsilon)n$. We note that Lemma 2 guarantees that with high probability at most $\frac{n}{2}$ vertices in R are at distance more than k^* from F and so with high probability $\beta_k \geq 1/2$ for $k \geq 0$.

We now move to showing that for sufficiently large k of size $O(\log\log n)$, a large fraction of the vertices are within distance k of the free vertices F with high probability.

Lemma 3. *Suppose that $d \geq 8$ and if $\varepsilon \leq \frac{1}{6}$ assume $d \geq 4 + 2\varepsilon - 2(1+\varepsilon)\log\left(\frac{\varepsilon}{1+\varepsilon}\right)$. Then with high probability $1 - \beta_k = O\left(\frac{\log^{\gamma_0} n}{(d-1)^k}\right)$ for k such that $1/2 \geq 1 - \beta_k \geq 2d^3 \log n/n$.*

The reader should not be put off by the fact that the lemma only has content for $k = \Omega(\log\log n)$. It is only needed for these values of k.

Proof. We must show that

$$1 - \beta_k = O\left(\frac{\log^{\gamma_0} n}{(d-1)^k}\right) \quad \text{whenever } 1 - \beta_k \geq 2d^3 \log n/n. \tag{14}$$

Assume that the high probability event in Lemma 1 occurs and $1 - \beta_k \geq \log n/n$ and the G-neighborhood X_k of Y_k in L has size at least $n - \frac{(1-\beta_k)n}{\alpha_k}$ where $\alpha_k = d - 1 - \frac{d + \log d}{1 - \log(1-\beta_k)}$. Note that for $\beta_k \geq \frac{3}{4}$ and $d \geq 8$ this implies

$$\alpha_k \geq \frac{d}{3} \text{ and } \frac{(d+\log d)/(d-1)}{1 - \log(1-\beta_k)} \leq 0.9. \tag{15}$$

First assume that $\beta_0 \geq 3/4$, we will deal with the case of $\beta_0 \geq \frac{1}{2}$ later. Note now that $Y_{k+1} = F \cup M(X_k)$ where $M(X_k) = \{y : (x,y) \in M \text{ for some } x \in X_k\}$. Thus $|Y_{k+1}| = (\beta_{k+1} + \varepsilon)n \geq \varepsilon n + n - \frac{(1-\beta_k)n}{\alpha_k}$. This implies that

$$\begin{aligned}
1 - \beta_{k+1} &\leq \frac{1-\beta_k}{\alpha_k} && (16)\\
&= \frac{1-\beta_k}{d-1}\left(1 - \frac{(d+\log d)/(d-1)}{1-\log(1-\beta_k)}\right)^{-1}\\
&\leq \frac{1-\beta_k}{d-1}\exp\left(h\left(\frac{(d+\log d)/(d-1)}{1-\log(1-\beta_k)}\right)\right) && (17)\\
&\leq \frac{1-\beta_k}{d-1}\exp\left(h\left(\frac{(d+\log d)/(d-1)}{1-\log(1-\beta_0)+k\log(d/3)}\right)\right). && (18)
\end{aligned}$$

In (17) we let $h(x) = x + 3x^2$ and note that $(1-x)^{-1} \leq \exp(h(x))$ for $x \in [0, .9]$. For (18) we have assumed that $1 - \beta_k \leq 3^k(1-\beta_0)/d^k$, which follows from (15) and (16) provided $\beta_k \geq 3/4$.

For $\beta \in [\frac{1}{2}, \frac{3}{4}]$ note that α_k is increasing in d and β. Also starting with $\beta_0 = \frac{1}{2}$ and using $d = 8$ we see numerically that $1 - \beta_3 \leq \frac{\frac{1}{2}}{\alpha_0\alpha_1\alpha_2} \leq \frac{1}{4}$. Thus after at most 3 steps we can assume $\beta \geq 3/4$. To simplify matters we will assume $\beta_0 \geq 3/4$, since doing this will only "shift" the indices by at most 3 and distort the

equations by a $O(1)$ factor.
Using inequality (18) repeatedly gives

$$\begin{aligned}
&1-\beta_{k+1} \leq \\
&\frac{1-\beta_0}{(d-1)^{k+1}} \exp\left(\frac{d+\log d}{(d-1)\log(d/3)} \sum_{j=0}^{k} \frac{1}{j+\frac{1-\log(1-\beta_0)}{\log(d/3)}} + O\left(\sum_{j=0}^{k} \frac{1}{\left(j+\frac{1-\log(1-\beta_0)}{\log(d/3)}\right)^2}\right)\right) \\
&\leq \frac{1-\beta_0}{(d-1)^{k+1}} \exp\left(\frac{d+\log d}{(d-1)\log(d/3)} \log\left(\frac{1-\log(1-\beta_k)}{1-\log(1-\beta_0)}\right) + O(1)\right) \qquad (19) \\
&\leq O\left(\frac{\log^{\gamma_0} n}{(d-1)^{k+1}}\right).
\end{aligned}$$

Note that (19) is obtained as follows:

$$\sum_{j=0}^{k} \frac{1}{j+\frac{1-\log(1-\beta_0)}{\log(d/3)}} = \log\left(\frac{k+\zeta}{\zeta}\right) + O(1) \leq \log\left(\frac{1-\log(1-\beta_k)}{1-\log(1-\beta_0)}\right) + O(1) \qquad (20)$$

where $\zeta = \frac{1-\log(1-\beta_0)}{\log(d/3)}$. Now $1-\beta_k \leq 3^k(1-\beta_0)/d^k$ implies that $k \leq \log((1-\beta_0)/(1-\beta_k))/\log(d/3)$. Substituting this upper bound for k into the middle term of (20) yields the right hand side.

We now require some additional structural lemmas regarding the graph in order to show that a breadth first search on the graph expands suitably.

Lemma 4. Whp *G does not contain a connected subgraph H on $2k+1$ vertices with $2k+3d$ edges, where $k+1$ vertices come from L and k vertices come from R for $k \leq \frac{1}{6}\log_d n$.*

Proof. We put an upper bound on the probability of the existence of such a subgraph using the union bound. Let the vertices of H be fixed, any such graph H can be constructed by taking a bipartite spanning tree on the $k+1$ and k vertices and adding j edges. Thus the probability of such a subgraph is at most

$$\begin{aligned}
&\binom{n}{k+1}\binom{(1+\varepsilon)n}{k} k^{(k+1)-1}(k+1)^{k-1}\left(k(k+1)\right)^j \left(\frac{j}{(1+\varepsilon)n}\right)^{2k+j} \\
&\qquad \leq \left(\frac{en}{k+1}\right)^{k+1}\left(\frac{e(1+\varepsilon)n}{k}\right)^k k^k(k+1)^{k-1}\left(\frac{jk(k+1)}{(1+\varepsilon)n}\right)^j j^{2k}\left(\frac{1}{(1+\varepsilon)n}\right)^{2k} \\
&\qquad \leq n\,(ej)^{2k}\left(\frac{jk(k+1)}{n}\right)^j \qquad (21)
\end{aligned}$$

For $j=3d$ and $k \geq \frac{1}{6}\log_d n$ we have $\frac{3dk(k+1)}{n} \leq n^{-1+\frac{1}{d}}$ and $(3ed)^{2k} \leq \exp\left(\log(d^3)\frac{\log n}{3\log d}\right) = n$ and (21) is at most $n^2 n^{-3d+2} = O(n^{-2d})$.

Lemma 5. Whp *there do not exist $S \subseteq L$, $T \subseteq R$ such that $N(S) \subseteq T$, $2d^2\log n \leq s = |S| \leq n/d$, $t = |T| \leq (d-1-\theta_s)s$ and $\theta_s = \frac{d+\log d}{\log(n/((d-1)s))} \geq \frac{d+\log d}{\log(n/t)}$.*

Proof. The expected number of pairs S, T satisfying (i),(ii) can be bounded by

$$\begin{aligned}
\sum_{s=2d^2\log n}^{n/d} \binom{n}{s}\binom{(1+\varepsilon)n}{t}\left(\frac{t}{(1+\varepsilon)n}\right)^{ds} &\leq \sum_{s=2d^2\log n}^{n/d} \left(\frac{ne}{s}\right)^s \left(\frac{(1+\varepsilon)ne}{t}\right)^t \left(\frac{t}{(1+\varepsilon)n}\right)^{ds} \\
&\leq \sum_{s=2d^2\log n}^{n/d} \left(\frac{ne}{s}\right)^s e^{(d-1-\theta_s)s}\left(\frac{t}{(1+\varepsilon)n}\right)^{ds-(d-1-\theta_s)s} \\
&\leq \sum_{s=2d^2\log n}^{n/d} \left(\frac{t}{s}\frac{e^{d-\theta_s}}{(1+\varepsilon)^{1+\theta_s}}\left(\frac{t}{n}\right)^{\theta_s}\right)^s \leq \sum_{s=2d^2\log n}^{n/d} \left((d-1)e^{d-\theta_s}\left(\frac{t}{n}\right)^{\theta_s}\right)^s
\end{aligned}$$

$$\leq \sum_{s=2d^2\log n}^{n/d}\left(\frac{d-1}{d}\right)^s = O\left(n^{-\frac{2d^2\log n}{d}}\right) = O\left(n^{-2d}\right)$$

4 Random Walks

Suppose now that we are in the process of adding u to the hash table. For our analysis, we consider exploring a subgraph of G using breadth-first search, starting with the root $u \in L$ and proceeding until we reach F. We emphasize that this is not the behavior of our algorithm; we merely need to establish some properties of the graph structure, and the natural way to do that is by considering a breadth-first search from u.

Let $L_1 = \{u\}$. Let the R-neighbors of x be $w_1, w_2, \ldots, w_d$ and suppose that none of them are in F. Let $R_1 = \{w_1, w_2, \ldots, w_d\}$. Let $L_2 = \{v_1, v_2, \ldots, v_d\}$ where v_i is matched with w_i in M, for $i = 1, 2, \ldots, d$. In general, suppose we have constructed L_k for some k. R_k consists of the R-neighbors of L_k that are not in $R_{\leq k-1} = R_1 \cup \cdots \cup R_{k-1}$ and L_{k+1} consists of the M-neighbors of R_i. An edge (x, y) from L_k to R is *wasted* if either (i) $y \in R_j, j < k$ or if there exists $x' \in L_k$, $x' < x$ such that the edge $(x', y) \in G$. We let

$$k_0 = \lfloor \log_{d-1}(n) - 1 \rfloor$$

and $\rho_k = |R_k|, \lambda_k = |L_k|$ for $1 \leq k \leq k_0$. Assume for the moment that

$$|R_k| \cap F = \emptyset \text{ for } 1 \leq k \leq k_0. \tag{22}$$

Lemma 6. *Assume that* (22) *holds. Then*

$$\rho_{k_0} = \Omega\left(\frac{n}{\log^{\gamma_1} n}\right). \tag{23}$$

Proof. We can assume that $1 - \beta_{k_0} \geq 2d^3 \log n/n$. If $1 \leq k \leq k_1 = \left\lfloor \frac{\log_d n}{6} \right\rfloor$ then Lemma 4 implies that we generate at most $3d$ wasted edge in the construction of $L_j, R_j, 1 \leq j \leq k$. If we consider the full BFS path tree, where vertices can be repeated, then each internal vertex of the tree L has $d-1$ children. For every wasted edge we cut off a subtree of the full BFS tree, what remains when all the wasted edges have been cut is the regular BFS tree. Clearly the worst case is when all the subtrees cut off are close to the root, thus $3d$ wasted edges can at most stunt the growth of the tree for 4 levels ($d-2$ edges cut at the 3 lowest levels and 6 edges cut off at the 4-th level). This means that

$$\rho_k > (d-1)^{k-5} \text{ for } 1 \leq k \leq k_1. \tag{24}$$

In particular $\rho_{k_1} = \Omega\left((d-1)^{\frac{\log_d n}{6}}\right) = \Omega\left(2d^2 \log n\right)$ so Lemma 5 applies to the BFS tree at this stage. In general Lemma 5 implies that for $j \geq k_1$

$$\rho_1 + \rho_2 + \cdots + \rho_j \geq (d - 1 - \theta_s)s \tag{25}$$

where

$$s = \lambda_1 + \lambda_2 + \cdots + \lambda_j = 1 + \rho_1 + \rho_2 + \cdots + \rho_{j-1}. \tag{26}$$

This follows from the fact that $\lambda_1 = 1$ and (22) implies $\lambda_j = \rho_{j-1}$ for $j \geq 2$.

Now $\lambda_j \leq (d-1)\lambda_{j-1}$ for $j \geq 3$ and so s in (26) satisfies $s \leq 1 + d + d(d-1) + \cdots + d(d-1)^{j-2} < (d-1)^{j-1}$. Thus θ_s in (25) satisfies

$$\theta_s \leq \phi_j = \frac{d + \log d}{\log n - j\log(d-1)}.$$

Thus, by (25) and (26) we have (after dropping a term)

$$\rho_j \geq (d - 2 - \phi_j)(\rho_1 + \rho_2 + \cdots + \rho_{j-1}). \tag{27}$$

An induction then shows that for $\ell \geq 1$,

$$\rho_{k_1+\ell} \geq (\rho_1 + \cdots + \rho_{k_1})(d - 2 - \phi_{k_1+\ell}) \prod_{k=1}^{\ell-1}(d - 1 - \phi_{k_1+k}). \tag{28}$$

Indeed the case $\ell = 1$ follows directly from (27). Then, by induction,

$$\rho_{k_1+\ell+1} \geq (\rho_1 + \cdots + \rho_{k_1})(d-2-\phi_{k_1+\ell+1}) \left(1 + \sum_{k=1}^{\ell}(d-2-\phi_{k_1+k}) \prod_{i=1}^{k-1}(d-1-\phi_{k_1+i})\right)$$
$$= (\rho_1 + \cdots + \rho_{k_1})(d-2-\phi_{k_1+\ell+1}) \prod_{k=1}^{\ell}(d-1-\phi_{k_1+k}). \tag{29}$$

To check (29) we can use induction. Assume that

$$1 + \sum_{k=2}^{\ell+1}(d-2-\phi_{k_1+k}) \prod_{i=2}^{k}(d-1-\phi_{k_1+i}) = \prod_{k=2}^{\ell+1}(d-1-\phi_{k_1+k})$$

and then multiply both sides by $d-1-\phi_{k_1+1}$.

We deduce from (24) and (28) that provided $k_1 + \ell \leq k_0$ (which implies $\frac{\phi_{k_1+\ell}}{d-1} \leq \frac{1}{2}$),

$$\rho_{k_1+\ell} \geq ((d-1)^{k_1-5} - 1)(d-2-\phi_{k_1+\ell}) \prod_{k=1}^{\ell-1}(d-1-\phi_{k_1+k})$$
$$\geq \frac{1}{2}(d-1)^{k_1+\ell-4} \exp\left\{-\frac{1}{d-1}\sum_{k=1}^{\ell}\phi_{k_1+k} - \frac{1}{(d-1)^2}\sum_{k=1}^{\ell}\phi_{k_1+k}^2\right\} \tag{30}$$

Note next that

$$\sum_{k=1}^{\ell}\phi_{k_1+k} = \frac{d+\log d}{\log(d-1)}\left(\log\left(\frac{\log n - k_1\log(d-1)}{\log n - (k_1+\ell)\log(d-1)}\right) + O(1)\right) \leq \frac{d+\log d}{\log(d-1)}(\log\log n + O(1))$$

and $\sum_{k=1}^{\ell}\phi_{k_1+k}^2 = O(1)$. Thus, putting $\ell = k_0 - k_1$ we get

$$\rho_{k_0} = \Omega\left(\frac{(d-1)^{k_0}}{(\log n)^{(d+\log d)/((d-1)\log(d-1))}}\right)$$

and the lemma follows.

5 Proof of Theorem 1

Let S denote the set of vertices $v \in R$ at distance at most $\Delta = k^* + (\gamma_0+\gamma_1)\log_{d-1}\log n + 2K$ from F, where K is a large constant and k^* is given in Lemma 2. Then by Lemma 3

$$|R \setminus S| \leq \frac{n}{(d-1)^K \log^{\gamma_1}(n)}.$$

We have used $(d-1)^K$ to "soak up" the hidden constant in the statement of Lemma 3 and the requirement $1-\beta_k \geq 2d^2\log n/n$ in Lemma 3 does not cause problems. If it fails then at most $O(\log n)$ vertices are at distance greater than Δ from F.

If K is sufficiently large then Lemma 6 implies that

$$|R \setminus S| \leq \rho_{k_0}/2. \tag{31}$$

Every vertex $v \in S$ has a path of length $l \leq \Delta$ to a free vertex and the probability that the random walk follows this path is $\left(\frac{1}{d-1}\right)^l \geq \left(\frac{1}{d-1}\right)^{\Delta}$, which is a lower bound on the probability the algorithm finds a free vertex within Δ steps, starting from $v \in S$. We now split the random walk into rounds, and each round into two phases.

The first phase starts when the round starts and ends when the random walk reaches a vertex of S or after k_0 steps(possibly the first phase is empty). Then, the second phase starts and ends either when the random walk reaches a free vertex or after Δ steps, finishing this round. The length of the first phase is at most k_0 and in the second phase takes at most Δ steps.

Claim. Starting from a vertex $v \notin S$ the expected number of rounds until the random walk is in S is at most $O(\log^{\gamma_1} n)$. Indeed the probability that a random walk of length k_0 passes through S is at least $\frac{\rho_{k_0} - |R \setminus S|}{(d-1)^{k_0}} = \Omega(\log^{-\gamma_1} n)$.

By Claim 5 we have a $\Omega(\log^{-\gamma_1} n)$ chance of reaching S at the end of the first phase. When we start the second phase we have at least a $\left(\frac{1}{d-1}\right)^{\Delta}$ probability of reaching a free vertex, thus ending the random walk. Then the number of rounds until we reach a free vertex is dominated by a geometric distribution with parameter $\Omega\left(\left(\frac{1}{d-1}\right)^{\Delta} \log^{-\gamma_1} n\right)$ and thus the expected number of rounds is $O((d-1)^{\Delta} \log^{\gamma_1} n)$. Since both Lemma 3 and Claim 5 apply regardless of the starting vertex, this shows that the expected number of steps until we reach a free vertex is at most

$$\begin{aligned} O\left(k_0 \log^{\gamma_1} n (d-1)^{\Delta}\right) &= O\left((\log n)\,(\log^{\gamma_1} n)(d-1)^{(\gamma_0+\gamma_1)\log_{d-1}\log n + O(1)}\right) \\ &= O\left(\log^{1+\gamma_0+2\gamma_1} n\right). \end{aligned}$$

There is still the matter of Assumption (22). This is easily dealt with. If we find $v \in R_k \cap F$ then we are of course delighted. So, we could just add a dummy tree extending $2(k_0 - k)$ levels from v where each vertex in the last level is in F. The conclusion of Claim 5 will remain unchanged. This completes the proof of Theorem 1.

6 Conclusion

We have demonstrated that for sufficiently large d with high probability the graph structure of the resulting cuckoo graph is such that, regardless of the staring vertex, the random-walk insertion method will reach a free vertex in polylogarithmic time with high probability. Obvious directions for improvement include reducing the value of d for which this type of result holds, and reducing the exponent in the time bound.

References

1. Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced Allocations. *SIAM Journal on Computing*, 29(1):180-200, 1999.
2. A. Broder and A. Karlin. Multilevel Adaptive Hashing. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms* (SODA), pp. 43-53, 1990.
3. A. Broder and M. Mitzenmacher. Using Multiple Hash Functions to Improve IP Lookups. *Proceedings of the 20th IEEE International Conference on Computer Communications* (INFOCOM), pp. 1454-1463, 2001.
4. L. Devroye and P. Morin. Cuckoo Hashing: Further Analysis. *Information Processing Letters*, 86(4):215-219, 2003.
5. M. Dietzfelbinger and C. Weidling. Balanced Allocation and Dictionaries with Tightly Packed Constant Size Bins. *Theoretical Computer Science*, 380(1-2):47-68, 2007.
6. D. Fotakis, R. Pagh, P. Sanders, and P. Spirakis. Space Efficient Hash Tables With Worst Case Constant Access Time. *Theory of Computing Systems*, 38(2):229-248, 2005.
7. A. Kirsch and M. Mitzenmacher. Using a Queue to De-amortize Cuckoo Hashing in Hardware. In *Proceedings of the Forty-Fifth Annual Allerton Conference on Communication, Control, and Computing*, 2007.
8. 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*, pp. 611-622, 2008.
9. A. Kirsch and M. Mitzenmacher. The Power of One Move: Hashing Schemes for Hardware. In *Proceedings of the 27th IEEE International Conference on Computer Communications* (INFOCOM), pp. 565-573, 2008.
10. R. Kutzelnigg. Bipartite Random Graphs and Cuckoo Hashing. In *Proceedings of the Fourth Colloquium on Mathematics and Computer Science*, 2006.
11. M. Mitzenmacher and S. Vadhan. Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA), pp. 746-755, 2008.
12. R. Pagh and F. Rodler. Cuckoo Hashing. *Journal of Algorithms*, 51(2):122-144, 2004.
13. B. Vöcking. How Asymmetry Helps Load Balancing. *Journal of the ACM*, 50(4):568-589, 2003.