Theory Comput. Systems **38**, 229–248 (2005)
DOI: 10.1007/s00224-004-1195-x

Theory of Computing Systems

Space Efficient Hash Tables with Worst Case Constant Access Time*

Dimitris Fotakis,[1] Rasmus Pagh,[2] Peter Sanders,[3] and Paul Spirakis[4]

[1] Department of Information and Communication Systems Engineering,
University of the Aegean, 83200 Karlovasi, Samos, Greece
fotakis@aegean.gr

[2] IT University of Copenhagen,
DK1017 Copenhagen K, Denmark
pagh@itu.dk

[3] Fakultät für Informatik, Universität Karlsruhe,
76128 Karlsruhe, Germany
sanders@ira.uka.de

[4] Computer Technology Institute,
26110 Patras, Greece
spirakis@cti.gr

Abstract. We generalize Cuckoo Hashing [23] to *d-ary Cuckoo Hashing* and show how this yields a simple hash table data structure that stores n elements in $(1 + \varepsilon)\, n$ memory cells, for any constant $\varepsilon > 0$. Assuming uniform hashing, accessing or deleting table entries takes at most $d = \mathrm{O}(\ln(1/\varepsilon))$ probes and the expected amortized insertion time is constant. This is the first dictionary that has worst case constant access time and expected constant update time, works with $(1+\varepsilon)\, n$ space, and supports satellite information. Experiments indicate that $d = 4$ probes suffice for $\varepsilon \approx 0.03$. We also describe variants of the data structure that allow the use of hash functions that can be evaluated in constant time.

* A preliminary version of this work appeared in the *Proceedings of the* 20*th Annual Symposium on Theoretical Aspects of Computer Science* (*STACS* 2003), Lecture Notes in Computer Science 2607. This work was partially supported by DFG Grant SA 933/1-1 and the Future and Emerging Technologies programme of the EU under Contract Number IST-1999-14186 (ALCOM-FT). The present work was initiated while the second author was at BRICS, Aarhus University, Denmark. Most of this work was done while the authors were at the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

1. Introduction

The efficiency of many programs crucially depends on hash table data structures, because they support constant expected access time. Also, hash table data structures that support *worst case* constant access time have been known for quite some time [12], [9]. Such worst case guarantees are relevant for real time systems and parallel algorithms where delays of a single processor could make all the others wait. A particularly fast and simple hash table with worst case constant access time is *Cuckoo Hashing* [23]: Each element is mapped to two tables t_1 and t_2 of size $(1+\varepsilon)\,n$ using two hash functions h_1 and h_2, for any $\varepsilon > 0$. A factor above two in space expansion is sufficient to ensure with high probability that each element e can be stored either in $t_1[h_1(e)]$ or $t_2[h_2(e)]$. The main trick is that insertion moves elements to different table positions to make room for the new element.

To our best knowledge, all previously known hash tables with worst case constant access time and sublinear insertion time share the drawback of a factor of at least two in memory blowup. In contrast, hash tables with only expected constant access time that are based on open addressing can work with memory consumption $(1+\varepsilon)\,n$. In the following, ε stands for an arbitrary positive constant.

The main contribution of this paper is a hash table data structure with worst case constant access time and memory consumption only $(1+\varepsilon)\,n$. The access time is $O(\ln(1/\varepsilon))$ which is in some sense optimal, and the expected insertion time is also constant. The proposed algorithm is a rather straightforward generalization of Cuckoo Hashing to *d-ary Cuckoo Hashing*: each element is stored at the position dictated by one out of d hash functions. In our analysis, insertion is performed by Breadth First Search (BFS) in the space of possible ways to make room for a new element. In order to ensure that the amount of memory used for bookkeeping in the BFS is negligible, we limit the number of nodes that can be searched to $o(n)$, and perform a rehash if this BFS does not find a way of accommodating the elements. For practical implementation, a random walk can be used. Unfortunately, the analysis that works for the original (binary) Cuckoo Hashing and $(\log n)$-wise independent hash functions [23] breaks down for $d \geq 3$. Therefore we develop new approaches and give an analysis of the simple algorithm outlined above for the case that hash functions are truly random. As observed by Dietzfelbinger [6], similar results can be obtained using an explicit family of constant time evaluable hash functions based on [21] or [10].

We also provide experimental evidence which indicates that d-ary Cuckoo Hashing is even better in practice than our theoretical performance guarantees. For example, at $d = 4$, we can achieve 97 % space utilization and at 90 % space utilization, insertion requires only about 20 memory probes on the average, i.e., only about a factor two more than for $d = \infty$.

We also present *Filter Hashing*, an alternative to d-ary Cuckoo Hashing that uses polynomial hash functions of degree $O(\sqrt{d})$. It has the same performance as d-ary Cuckoo Hashing except that it uses $d = O(\ln^2(1/\varepsilon))$ probes for an access in the worst case.

A novel feature of both d-ary Cuckoo Hashing (in the variant implemented for the experiments) and Filter Hashing is that we use several small hash tables with only a single possible location for each element and table. Traditional hashing schemes use large hash

tables where good space utilization is achieved by having many possible locations for each element.

1.1. *Related Work*

Space Efficient Dictionaries. A *dictionary* is a data structure that stores a set of elements, and associates some piece of information with each element. Given an element, a dictionary can look up whether it is in the set, and, if so, return its associated information. Usually elements come from some universe of bounded size. If the universe has size m, the information theoretical lower bound on the number of bits needed to represent a set of n elements (without associated information) is $B = n \log(em/n) - \Theta(n^2/m) - \mathrm{O}(\log n)$. This is roughly $n \log n$ bits less than, say, a sorted list of elements. If $\log m$ is large compared with $\log n$, using n words of $\log m$ bits is close to optimal.

A number of papers have given data structures for storing sets in near-optimal space, while supporting efficient lookups of elements, and other operations. Cleary [4] showed how to implement a variant of linear probing in space $(1 + \varepsilon)B + \mathrm{O}(n)$ bits, under the assumption that a truly random permutation on the key space is available. The expected average time for (failed) lookups and insertions is $\mathrm{O}(1/\varepsilon^2)$, as in ordinary linear probing. A space usage of $B + \mathrm{o}(n) + \mathrm{O}(\log \log m)$ bits was obtained in [22] for the *static* case. Both these data structures support associated information using essentially optimal additional space.

Other works have focused on dictionaries *without* associated information. Brodnik and Munro [3] achieve space $\mathrm{O}(B)$ in a dictionary that has worst case constant lookup time and amortized expected constant time for insertions and deletions. The space usage was recently improved to $B + o(B)$ bits by Raman and Rao [24]. Since these data structures are not based on hash tables, it is not clear that they extend to support associated information. In fact, Raman and Rao mention this extension as a goal of future research.

Our generalization of Cuckoo Hashing uses a hash table with $(1 + \varepsilon)\, n$ entries of $\log m$ bits. As we use a hash table, it is trivial to store associated information along with elements. The time analysis depends on the hash functions used being truly random. For many practical hash functions, the space usage can be decreased to $(1 + \varepsilon)\, B + \mathrm{O}(n)$ bits using *quotienting* (as in [4], [22]). Thus, our scheme can be seen as an improvement of the result of Cleary to worst case lookup bounds (even having a better dependence on ε than his average case bounds). However, there remains a gap between our experimental results for insertion time and our theoretical upper bound, which does not beat Cleary's.

Open Addressing Schemes. Cuckoo Hashing falls into the class of open addressing schemes, as it places keys in a hash table according to a sequence of hash functions. The worst case $\mathrm{O}(\ln(1/\varepsilon))$ bound on lookup time matches the *average* case bound of classical open addressing schemes like double hashing. Yao [29] showed that this bound is the best possible among all open addressing schemes that do not move elements around in the table. A number of hashing schemes move elements around in order to improve or remove the dependence on ε in the average lookup time [1], [13], [16], [17], [25].

In the classical open addressing schemes some elements can have a retrieval cost of $\Omega(\log n / \log \log n)$. Bounding the worst case retrieval cost in open addressing schemes was investigated by Rivest [25], who gave a polynomial time algorithm for arranging

keys so as to minimize the worst case lookup time. However, no bound was shown on the expected worst case lookup time achieved. Rivest also considered the dynamic case, but the proposed insertion algorithm was only shown to be expected constant time for low load factors (in particular, nothing was shown for $\varepsilon \leq 1$).

Matchings in Random Graphs. Our analysis uses ideas from two seemingly unrelated areas that are connected to Cuckoo Hashing by the fact that all three problems can be understood as finding matchings in some kind of random bipartite graphs.

The proof that space consumption is low is similar in structure to the result in [28] and [27] that two hash functions suffice to map n elements (disk blocks) to D places (disks) such that no disk gets more than $\lceil n/D \rceil + 1$ blocks. The proof details are quite different however. In particular, we derive an analytic expression for the relation between ε and d. Similar calculations may help to develop an analytical relation that explains for which values of n and D the "+1" in $\lceil n/D \rceil + 1$ can be dropped. In [27] this relation was only tabulated for small values of n/D.

The analysis of insertion time uses expansion properties of random bipartite graphs. Motwani [19] uses expansion properties to show that the algorithm by Hopcroft and Karp [14] finds perfect matchings in random bipartite graphs with $m > n \ln n$ edges in expected time $O(m \log n / \log \log n)$. He shows an $O(m \log n / \log d)$ bound for the *d-out* model of random bipartite graphs, where all nodes are constrained to have degree at least $d \geq 4$.

Our analysis of insertion can be understood as an analysis of a simple incremental algorithm for finding perfect matchings in a random bipartite graph where n nodes on the left side are constrained to have constant degree d whereas there are $(1 + \varepsilon) n$ nodes on the right side without a constraint on the degree. We feel that this is a more natural model for sparse graphs than the d-out model because there seem to be many applications where there is an asymmetry between the two node sets and it is unrealistic to assume a lower bound on the degree of a right node (e.g., [28], [26], and [27]).

Under these conditions we get *linear* run time even for very sparse graphs using a very simple algorithm that has the additional advantage of allowing incremental addition of nodes. The main new ingredient in our analysis is that besides expansion properties, we also prove *shrinking properties* of nodes not reached by a BFS. An aspect that makes our proof more difficult than the case in [19] is that our graphs have weaker expansion properties because they are less dense (or less regular for the d-out model).

1.2. *Overview*

In Section 2 we introduce Cuckoo Hashing as a matching problem in a class of random bipartite graphs. Section 3 constitutes the main part of the paper. In Section 3.1 we prove that for $d \geq 2(1 + \varepsilon) \ln(e/\varepsilon)$ and truly random hash functions, d-ary Cuckoo Hashing results in bipartite graphs which contain a matching covering all left vertices/elements with high probability (henceforth "whp"[1]). In Section 3.2 we show that for somewhat larger values of d, an incremental algorithm which augments along a shortest augmenting path takes $(1/\varepsilon)^{O(\ln d)}$ expected time per element and visits at most $o(n)$ vertices

[1] In this paper "whp" means "with probability at least $1 - O(1/n)$."

before it finds an augmenting path whp. Section 4 complements the theoretical analysis with experiments which justify the practical efficiency of d-ary Cuckoo Hashing. Filter Hashing is presented and analyzed in Section 5. We show that for $d = \Theta(\ln^2(1/\varepsilon))$ and polynomial hash functions of degree $O(\ln(1/\varepsilon))$, Filter Hashing stores almost all elements (e.g., at least $(1 - \varepsilon/4)n$ of them) in n memory cells whp. Some directions for further research and a modification of d-ary Cuckoo Hashing for hash functions that can be evaluated in constant time are discussed in Section 6.

2. Preliminaries

A natural way to define and analyze d-ary Cuckoo Hashing and Filter Hashing is through matchings in random bipartite graphs. The elements can be thought of as the left vertices and the memory cells as the right vertices of a bipartite graph $B(L, R, E)$. The edges of B are determined by the hash functions. An edge connecting a left vertex to a right vertex indicates that the corresponding element can be stored in the corresponding memory cell. Any one-to-one mapping of elements/left vertices to memory cells/right vertices forms a matching in B. Since every element is stored in some cell, this matching is L-perfect, i.e., it covers all the left vertices of B.

Having fixed an L-perfect matching M, we can think of B as a directed graph, where the edges in $E \backslash M$ are directed from left to right and the edges in M are directed from right to left (Figure 1). Therefore, each right vertex has outdegree at most one. The set of *free vertices* $F \subseteq R$ simply consists of the right vertices with no outgoing edges (i.e., zero outdegree). Then any directed path from an unmatched left vertex v to F is an augmenting path for M, because if we reverse the edge directions along such a path, we obtain a new matching which also covers v.

We always use X to denote a set of left vertices and Y to denote a set of right vertices. For a set of left vertices X, let $M(X)$ denote the set of X's mates according to the current matching M. For a set of vertices $S \subseteq L \cup R$, let $\Gamma(S)$ denote the neighborhood of S, i.e., $\Gamma(S) = \{v\colon \{u, v\} \in E, u \in S\}$. We should emphasize that whenever we use $\Gamma(S)$ or refer to the neighborhood of a vertex set, we consider the edges as undirected.

In the analysis of d-ary Cuckoo Hashing, we repeatedly use the following upper bound on binomial coefficients.

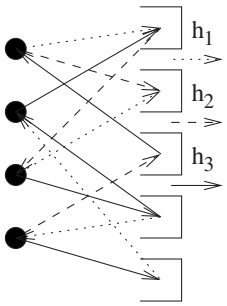

Fig. 1. Ternary Cuckoo Hashing ($d = 3$).

Proposition 1. *Given an integer n, for any integer k, $1 \le k \le n$,*

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

Proof. For completeness, we give a simple proof communicated to us by Dietzfelbinger [6]. Let $\mu = k/n \in (0, 1]$. We observe that

$$\begin{aligned}\binom{n}{k}\mu^k(1-\mu)^{n-k} &\le \sum_{j=0}^{n}\binom{n}{j}\mu^j(1-\mu)^{n-j}\\ &= (\mu + (1-\mu))^n = 1,\end{aligned}$$

which implies the proposition. □

3. *d*-Ary Cuckoo Hashing

In d-ary Cuckoo Hashing, we consider n elements stored in $(1+\varepsilon)n$ memory cells. Each element is stored in one out of d cells chosen uniformly at random and independently with replacement. Consequently, the resulting bipartite graph $B(L, R, E)$ has $|L| = n$ and $|R| = (1+\varepsilon)n$. Each left vertex has exactly d neighbors selected uniformly at random and independently (with replacement) from R.

3.1. *Existence of an L-Perfect Matching*

We start by proving that for sufficiently large values of d, such bipartite graphs contain an L-perfect matching whp.

Lemma 1. *Given a constant $\varepsilon \in (0, 1)$, for any integer $d \ge 2(1+\varepsilon)\ln(e/\varepsilon)$, the bipartite graph $B(L, R, E)$ contains an L-perfect matching with probability at least $1 - O(n^{4-2d})$.*

Proof. We establish a bound of $O(n^{4-2d})$ on the probability that there exists a set of left vertices X having less than $|X|$ neighbors. Then the lemma follows from Hall's theorem (e.g., Chapter 2 of [5]), which states that a bipartite graph contains an L-perfect matching if and only if any $X \subseteq L$ has at least $|X|$ neighbors.

For each integer k, $2 \le k \le n$, let $P(k)$ be the probability that there exists a set of k left vertices with at most k neighbors. The probability that the bipartite graph B does not contain an L-perfect matching is bounded by the sum of the probabilities $P(k)$ over all integers $2 \le k \le n$.

For an integer k, we fix a set of left vertices X and a set of right vertices Y both of size k. The probability that Y includes all neighbors of X is $(k/(1+\varepsilon)n)^{dk}$. Multiplying by the number of different sets X and Y, we obtain that

$$P(k) \le \binom{n}{k}\binom{(1+\varepsilon)n}{k}\left(\frac{k}{(1+\varepsilon)n}\right)^{dk}.$$

For $k = 2$, the probability that there exists a pair of left vertices with a single neighbor in R is $O(n^{3-2d})$. Let c be a sufficiently large constant. For $k \leq n/c$, using $\binom{n}{k} \leq (en/k)^k$, we obtain that $P(k) \leq e^{2k}k^{(d-2)k}/n^{(d-2)k}$. If $c > e^{d/(d-2)}$, the right-hand side of the inequality above is a non-increasing function of k and cannot exceed $O(n^{3-2d})$, for any $d \geq 3$. Therefore, for any $3 \leq k \leq n/c$, $P(k) = O(n^{3-2d})$.

For $n/c < k \leq n$, let $\mu = k/n \in (0, 1]$. Using Proposition 1, we obtain the following bound on the probability $P(\mu n)$:

$$P(\mu n) \leq \left[\left(\frac{1}{1-\mu}\right)^{1-\mu}\left(\frac{1}{\mu}\right)^{\mu}\left(\frac{1+\varepsilon}{1+\varepsilon-\mu}\right)^{1+\varepsilon-\mu}\left(\frac{1+\varepsilon}{\mu}\right)^{\mu}\left(\frac{\mu}{1+\varepsilon}\right)^{d\mu}\right]^n.$$

The value of d should make the quantity in square brackets strictly smaller than 1. Solving the resulting inequality, we obtain the following lower bound on d:

$$d > 1 + \frac{\mu\ln(1/\mu) + (1-\mu)\ln(1/(1-\mu)) + (1+\varepsilon-\mu)\ln((1+\varepsilon)/(1+\varepsilon-\mu))}{\mu\ln((1+\varepsilon)/\mu)}. \quad (1)$$

To simplify inequality (1), we observe that:

1. For all $\varepsilon \in (0, 1)$ and $\mu \in (0, 1]$, $\mu\ln(1/\mu) < \mu\ln((1+\varepsilon)/\mu)$.
2. For any fixed value of $\varepsilon \in (0, 1)$, $((1+\varepsilon-\mu)\ln((1+\varepsilon)/(1+\varepsilon-\mu))/(\mu\ln((1+\varepsilon)/\mu))$ is a non-decreasing function of μ in the interval $(0, 1]$. Hence, it is maximized for $\mu = 1$ achieving the value of $(\varepsilon\ln((1+\varepsilon)/\varepsilon))/\ln(1+\varepsilon)$.

We also need the following proposition.

Proposition 2. *For all* $\varepsilon \in (0, 1)$ *and* $\mu \in (0, 1]$,

$$\frac{(1-\mu)\ln(1/(1-\mu))}{\mu\ln((1+\varepsilon)/\mu)} \leq \frac{\varepsilon\ln((1+\varepsilon)/\varepsilon)}{\ln(1+\varepsilon)}.$$

Proof. For any fixed $\varepsilon \in (0, 1)$, we distinguish between $\mu \in (0, 1/(1+\varepsilon)]$ and $\mu \in (1/(1+\varepsilon), 1]$.

Case A. Let $\mu \in (0, 1/(1+\varepsilon)]$. Then

$$\frac{(1-\mu)\ln(1/(1-\mu))}{\mu\ln((1+\varepsilon)/\mu)} \leq \frac{(1-\mu)\ln(1/(1-\mu))}{\mu\ln(1/\mu)} = g_1(\mu).$$

The function $g_1(\mu)$ is strictly increasing with μ and

$$g_1\left(\frac{1}{1+\varepsilon}\right) = \frac{\varepsilon\ln((1+\varepsilon)/\varepsilon)}{\ln(1+\varepsilon)}.$$

Case B. Let $\mu \in (1/(1+\varepsilon), 1]$. Then

$$\frac{(1-\mu)\ln(1/(1-\mu))}{\mu\ln((1+\varepsilon)/\mu)} \leq \frac{(1-\mu)\ln(1/(1-\mu))}{\mu\ln(1+\varepsilon)} = g_2(\mu).$$

The function $g_2(\mu)$ is strictly decreasing with μ and

$$\gamma_2\left(\frac{1}{1+\varepsilon}\right) = \frac{\varepsilon \ln((1+\varepsilon)/\varepsilon)}{\ln(1+\varepsilon)}.$$ □

Consequently, for every integer $d \geq 2 + (2\varepsilon \ln((1+\varepsilon)/\varepsilon))/\ln(1+\varepsilon))$, the bipartite graph B contains an L-perfect matching with probability at least $1 - O(n^{4-2d})$. A brief calculation using $\ln(1+\varepsilon) > \varepsilon - \varepsilon^2/2$ shows that the above inequality is satisfied whenever $d \geq 2(1+\varepsilon)\ln(e/\varepsilon)$. □

By directly applying inequality (1) for specific values of ε, we obtain that if $\varepsilon \geq 0.57$ and $d = 3$, if $\varepsilon \geq 0.19$ and $d = 4$, and if $\varepsilon \geq 0.078$ and $d = 5$, the bipartite graph B contains an L-perfect matching whp. The experiments in Section 4 indicate that for $d = 3, 4, 5$, even smaller values of ε are possible.

We also show that the bound of Lemma 1 on d is essentially best possible.

Lemma 2. *If $d < (1+\varepsilon)\ln(1/\varepsilon)$ then $B(L, R, E)$ does not contain a perfect matching whp.*

Proof. We can think of the right vertices as bins and the edges as balls. Each ball is placed in a bin selected uniformly at random and independently. We have exactly $(1+\varepsilon)n$ bins and dn balls. The expected number of empty bins/isolated right vertices approaches $(1+\varepsilon)n\,e^{-d/(1+\varepsilon)}$ as n goes to infinity (e.g., Chapter 3 of [20].). Let β be a positive constant. For n larger than a suitable constant, if d is smaller than $(1+\varepsilon)\ln((1+\varepsilon)/(1+\beta)\varepsilon)$, the expected number of isolated right vertices is greater than $(1+\beta/2)\varepsilon n$. Furthermore, since the events "the right vertex v is isolated" are negatively associated (e.g., [11]), we can apply the Chernoff bound (e.g., Chapter 4 of [20]) and show that if $d < (1+\varepsilon)\ln((1+\varepsilon)/(1+\beta)\varepsilon)$, the number of isolated right vertices is greater than εn whp. Clearly, a bipartite graph B with more than εn isolated right vertices cannot contain an L-perfect matching. The lemma is obtained by setting $\beta = \varepsilon$. □

3.2. *The Insertion Algorithm*

To formulate and analyze the insertion algorithm for d-ary Cuckoo Hashing, we assume that the left vertices of the bipartite graph B arrive (along with their d random choices/edges) one-by-one in an arbitrary order and the algorithm incrementally maintains an L-perfect matching in B.

Let M be an L-perfect matching fixed before a left vertex v arrives. We recall that the edges in M are considered to be directed from right to left, while the edges not in M are directed from left to right. All the edges of v are initially directed from left to right, because v is not matched by M. We also recall that any directed path from v to the set of free vertices F is an augmenting path for M.

The insertion algorithm we analyze always augments the current matching along a shortest directed path from v to F. Such a path can be found by the equivalent of a Breadth First Search (BFS) in the directed version of B, which is implicitly represented by the d hash functions and the storage table. We ensure space efficiency by restricting the number of right vertices the BFS can visit to $o(n)$.

To avoid having to deal with dependencies among the random choices of a newly arrived left vertex and the current matching, we restrict our attention to the case where a left vertex that has been deleted from the hash table cannot be re-inserted.[2] The remaining section is devoted to the proof of the following theorem.

Theorem 1. *For every positive $\varepsilon < \frac{1}{5}$ and integer $d \geq 5 + 3\ln(1/\varepsilon)$, the incremental algorithm that augments along a shortest augmenting path takes $(1/\varepsilon)^{O(\ln d)}$ expected time per left vertex/element to maintain an L-perfect matching in B. Moreover, the algorithm visits at most $o(n)$ right vertices before it finds an augmenting path whp.*

Remark. Using the same techniques, we can prove that Theorem 1 holds for any $\varepsilon \in (0, 1)$. For ease of exposition, we restrict our attention to the most interesting case of small ε.

Augmentation Distance. In the proof of Theorem 1, we measure the distance from a vertex v to the set of free vertices F by only accounting for the number of left to right edges (*free edges* for short), or, equivalently, the number of left vertices appearing in a shortest path (respecting the edge directions) from v to F. We refer to this distance as the *augmentation distance* of v. We should emphasize that the augmentation distance of a vertex depends on the current matching M. The augmentation distance of a vertex v essentially measures the depth at which a BFS starting from v reaches F for the first time. Therefore, if a new left vertex has a neighbor at augmentation distance λ, a shortest augmenting path can be found in $O(d^{\lambda+1})$ time and space. To establish Theorem 1, we bound the number of right vertices at large augmentation distance.

Outline. The proof of Theorem 1 is divided into three parts. We first prove that the number of right vertices at augmentation distance at most λ grows exponentially with λ, whp, until almost half of the right vertices have been reached. We call this the *expansion property*. We next prove that for the remaining right vertices, the number of right vertices at augmentation distance greater than λ decreases exponentially with λ, whp. We call this the *shrinking property*. The proofs of both the expansion property and the shrinking property are based on the fact that for an appropriate choice of d, d-ary Cuckoo Hashing results in bipartite graphs which are good expanders, whp. Finally, we put the expansion property and the shrinking property together to show that the number of right vertices encountered by a BFS before an augmenting path is found is at most $o(n)$ whp, and the expected insertion time per element is constant.

Notation. For an integer λ, let Y_λ denote the set of right vertices at augmentation distance at most λ, and let X_λ denote the set of left vertices at augmentation distance at most λ. The sets X_λ and Y_λ can be computed inductively starting from the set of free vertices F. For $\lambda = 0$, $Y_0 = F$ and $X_0 = \emptyset$. For any integer $\lambda \geq 1$, we have $X_{\lambda+1} = \Gamma(Y_\lambda)$ and $Y_{\lambda+1} = M(X_{\lambda+1}) \cup F$.

[2] This restriction on deletions is easily overcome by just *marking* deleted elements, and only removing them when periodically rebuilding the hash table with new hash functions.

The Expansion Property. We first prove that if d is chosen appropriately large, every set of right vertices Y of size in the interval $[\varepsilon n, 3n/8]$ expands by a factor no less than $\frac{4}{3}$ whp (Lemma 3). This implies that the number of right vertices at augmentation distance at most λ is at least $(1+\frac{4}{3}^{\lambda})\varepsilon n$, as long as λ is so small that this number does not exceed $(\frac{1}{2}+\varepsilon)n$ (Lemma 5).

Lemma 3. *Given a constant $\varepsilon \in (0, \frac{1}{5})$, let $d \geq 5 + 3\ln(1/\varepsilon)$ be an integer. Then the probability that every set of right vertices Y, $\varepsilon n \leq |Y| \leq 3n/8$, has at least $4|Y|/3$ neighbors is at least $1 - 2^{-\Omega(n)}$.*

Proof. We first establish the following lemma whose proof is similar to the proof of Lemma 1.

Lemma 4. *Given a constant $\varepsilon \in (0, \frac{1}{5})$, let δ be any positive constant not exceeding $4(1-4\varepsilon)/(1+4\varepsilon)$, and let d be any integer such that*

$$d \geq 3 + 2\varepsilon + 2\delta(1+\varepsilon) + \frac{(2+\delta)\varepsilon \ln((1+\varepsilon)/\varepsilon)}{\ln(1+\varepsilon)}.$$

Then the probability that every set of left vertices X, $n/2 \leq |X| \leq (1-(1+\delta)\varepsilon)n$, has at least $(1+\varepsilon)n - (n-|X|)/(1+x\delta)$ neighbors is at least $1 - 2^{-\Omega(n)}$.

Proof. Let μ be any number in the interval $[\varepsilon, 1/2(1+\delta)]$ such that $(1-(1+\delta)\mu)n$ is an integer. For simplicity of presentation and without loss of generality, we assume that $(1+\varepsilon-\mu)n$ is also an integer. Let $P(\mu)$ be the probability that the bipartite graph B contains a set of left vertices X of size $(1-(1+\delta)\mu)n$ with at most $(1+\varepsilon-\mu)n$ neighbors in R. We bound the probability that the graph B does not satisfy the conclusion of the lemma by the sum of the probabilities $P(\mu)$ over all the different values of μ for which $(1-(1+\delta)\mu)n$ is an integer.

For a fixed value of μ, we fix a set of left vertices X of size $(1-(1+\delta)\mu)n$ and a set of right vertices Y of size $(1+\varepsilon-\mu)n$. The probability that all neighbors of X are included in Y is $((1+\varepsilon-\mu)/(1+\varepsilon))^{d(1-(1+\delta)\mu)n}$. Multiplying by the number of different sets X and Y, we obtain the following upper bound on $P(\mu)$:

$$P(\mu) \leq \binom{n}{(1+\delta)\mu n}\binom{(1+\varepsilon)n}{\mu n}\left(\frac{1+\varepsilon-\mu}{1+\varepsilon}\right)^{d(1-\mu(1+\delta))n}.$$

Using Proposition 1 and working as in the proof of Lemma 1, we obtain the following lower bound on d:

$$d > \frac{(1-\mu(1+\delta))\ln\left(\frac{1}{1-(1+\delta)\mu}\right) + (1+\delta)\mu\ln\left(\frac{1}{(1+\delta)\mu}\right)}{(1-\mu(1+\delta))\ln\left(\frac{1+\varepsilon}{1+\varepsilon-\mu}\right)}$$
$$+ \frac{(1+\varepsilon-\mu)\ln\left(\frac{1+\varepsilon}{1+\varepsilon-\mu}\right) + \mu\ln\left(\frac{1+\varepsilon}{\mu}\right)}{(1-\mu(1+\delta))\ln\left(\frac{1+\varepsilon}{1+\varepsilon-\mu}\right)}.$$

For all $\varepsilon \in (0, \frac{1}{5})$ and $\delta \in (0, 4(1-4\varepsilon)/(1+4\varepsilon)]$, the right-hand side of the above inequality is maximized for $\mu = \varepsilon$ yielding the following lower bound on d:

$$d > \frac{(1-\varepsilon(1+\delta))\ln\left(\frac{1}{1-(1+\delta)\varepsilon}\right) + (1+\delta)\varepsilon\ln\left(\frac{1}{(1+\delta)\varepsilon}\right)}{(1-\varepsilon(1+\delta))\ln(1+\varepsilon)}$$
$$+ \frac{\ln(1+\varepsilon) + \varepsilon\ln\left(\frac{1+\varepsilon}{\varepsilon}\right)}{(1-\varepsilon(1+\delta))\ln(1+\varepsilon)}. \tag{2}$$

The right-hand side of inequality (2) can be simplified by observing that

$$\ln\left(\frac{1}{(1+\delta)\varepsilon}\right) < \ln\left(\frac{1+\varepsilon}{\varepsilon}\right).$$

In addition, for all $\delta \in (0, 4(1-4\varepsilon)/1+4\varepsilon]$,

$$\frac{1}{1-(1+\delta)\varepsilon} \leq 1+2\varepsilon(1+\delta), \quad \text{and} \quad \frac{\ln\left(\frac{1}{1-(1+\delta)\varepsilon}\right)}{\ln(1+\varepsilon)} \leq 2(1+\delta).$$

Since $\mu \geq \varepsilon$ and there are at most n different values of μ to consider, the probability that the graph B does not have the claimed property is at least $1 - O(n\beta^n)$, for some constant $\beta < 1$ depending on the choice of d. □

Then we show that every bipartite graph B satisfying the conclusion of Lemma 4 also satisfies the conclusion of Lemma 3. To reach a contradiction, we assume that for some $\mu \in [\varepsilon, 1/2(1+\delta)]$, there is a set $Y \subseteq R$ of size μn with less than $(1+\delta)\mu n$ neighbors in L. Let X be the set of left vertices not included in the neighborhood of Y. Then X consists of more than $(1-(1+\delta)\mu)n$ vertices. By Lemma 4, X must have more than $(1+\varepsilon-\mu)n$ neighbors in R, which implies that some vertices of Y have neighbors in X, a contradiction to the definition of X.

To conclude the proof of Lemma 3, we observe that for $\varepsilon < \frac{1}{5}$, we can take $\delta = \frac{1}{3}$. Using the fact that $\ln(1+\varepsilon) > \varepsilon - \varepsilon^2/2$, the requirement of inequality (2) on d can be seen to be satisfied if $d \geq 5 + 3\ln(1/\varepsilon)$. □

The following lemma concludes the proof of the expansion property.

Lemma 5. *Let $B(L, R, E)$ be a bipartite graph satisfying the conclusion of Lemma 3. Then, for every integer λ, $1 \leq \lambda \leq \log_{4/3}(1/2\varepsilon)$, the number of right vertices at augmentation distance at most λ is at least $(1 + (\frac{4}{3})^\lambda)\varepsilon n$.*

Proof. We prove the lemma by induction on λ. We recall that Y_λ denotes the set of right vertices at augmentation distance at most λ. Since the bipartite graph B satisfies the conclusion of Lemma 3, every set of right vertices Y, $\varepsilon n \leq |Y| \leq 3n/8$, expands by a factor no less than $\frac{4}{3}$. The lemma holds for $\lambda = 1$, because $|Y_0| = |F| \geq \varepsilon n$ and, hence, at least $(1 + \frac{4}{3})\varepsilon n$ right vertices are included in Y_1.

For some integer λ, $1 \leq \lambda \leq \log_{4/3}(1/2\varepsilon) - 1$, let $|Y_\lambda| \geq (1 + (\frac{4}{3})^\lambda)\varepsilon n$. Then the neighborhood of Y_λ includes at least $(\frac{4}{3})^{\lambda+1}\,\varepsilon n$ left vertices. Since both the mates of these left vertices and the free vertices are at augmentation distance at most $\lambda + 1$, there are at least $(1 + (\frac{4}{3})^{\lambda+1})\varepsilon n$ right vertices at augmentation distance no greater than $\lambda + 1$, i.e., $|Y_{\lambda+1}| \geq (1 + (\frac{4}{3})^{\lambda+1})\varepsilon n$.

This argument works as long as $\lambda \leq \log_{4/3}(1/2\varepsilon)$, because the expansion argument works until the size of Y_λ becomes larger than $(\frac{1}{2} + \varepsilon)n$ for the first time. □

The Shrinking Property. By the expansion property, nearly half of the right vertices are at augmentation distance no greater than $\lambda^* = \lceil \log_{4/3}(1/2\varepsilon) \rceil$. Next, we show is that every set of left vertices X, $|X| \leq n/4$, has at least $2|X|$ neighbors in R whp (Lemma 7). This implies that the number of right vertices at augmentation distance greater than $\lambda^* + \lambda$ is at most $2^{-(\lambda+1)}\,n$ (Lemma 8).

Lemma 6. *Let $d \geq 8$ be an integer. Then the probability that every set of right vertices Y with $|Y| \geq (\frac{1}{2}+\varepsilon)n$ has at least $n-((1+\varepsilon)n-|Y|)/2$ neighbors is at least $1-\mathrm{O}(n^{4-d})$.*

Proof. We first show the following lemma whose proof is similar to the proof of Lemma 1.

Lemma 7. *Let γ be a positive constant, and let $d \geq (1+\log e)(2+\gamma)+\log(1+\gamma)$ be an integer. Then the probability that every set of left vertices X with $|X| \leq n/2(1+\gamma)$ has at least $(1+\gamma)|X|$ neighbors is at least $1 - \mathrm{O}(n^{3+\gamma-d})$.*

Proof. For a fixed integer k, $1 \leq k \leq n/2(1+\gamma)$, let $P(k)$ be the probability that there exists a set of left vertices of size k which does not expand by $(1+\gamma)$. Then

$$P(k) \leq \binom{n}{k}\binom{(1+\varepsilon)n}{(1+\gamma)k}\left(\frac{(1+\gamma)k}{(1+\varepsilon)n}\right)^{dk}.$$

Working similarly to the proof of Lemma 1, we bound $P(k)$ by $\mathrm{O}(n^{2+\gamma-d})$ for small values of k, e.g., $1 \leq k \leq n/c$, where c is a sufficiently large positive constant. For $k > n/c$, let $\mu = k/n \in (0, 1/2(1+\gamma)]$. Using Proposition 1, we obtain the following lower bound on d:

$$d > 1 + \gamma + \frac{\mu \ln\left(\frac{1}{\mu}\right) + (1-\mu)\ln\left(\frac{1}{1-\mu}\right)}{\mu \ln\left(\frac{1+\varepsilon}{(1+\gamma)\mu}\right)} + \frac{(1+\varepsilon-(1+\gamma)\mu)\ln\left(\frac{1+\varepsilon}{1+\varepsilon-(1+\gamma)\mu}\right)}{\mu \ln\left(\frac{1+\varepsilon}{(1+\gamma)\mu}\right)}. \tag{3}$$

We observe that the right-hand side of inequality (3) is maximized for $\mu = 1/2(1+\gamma)$ and cannot exceed $(1+\log e)(2+\gamma)+\log(1+\gamma)$. □

We observe that for $\gamma = 1$, inequality (3) is satisfied whenever $d \geq 8$. To conclude the proof of Lemma 6, we show that any graph B satisfying the conclusion of Lemma 7 also satisfies the conclusion of Lemma 6. To reach a contradiction, we assume that for some integer k, $0 \leq k \leq n/4$, there is a set of right vertices Y of size $(1+\varepsilon)n - 2k$ with less than $n-k$ neighbors. Let X be the set of left vertices not included in the neighborhood of Y. The size of X must be greater than k. By Lemma 7, X has more than $2k$ neighbors in R, which implies that some vertices of Y have neighbors in X, a contradiction. □

We observe that for every $\varepsilon \in (0, \frac{1}{5})$, $5 + 3\ln(1/\varepsilon) \geq 8$ and the hypothesis of Lemma 6 is satisfied by any choice of d satisfying the hypothesis of Theorem 1.

Lemma 8. *Let $B(L, R, E)$ be a bipartite graph satisfying the conclusions of Lemmas* 3 *and* 6 *and let $\lambda^* = \lceil \log_{4/3}(1/2\varepsilon) \rceil$. Then, for every integer $\lambda \geq 0$, the number of right vertices at augmentation distance greater than $\lambda + \lambda^*$ is at most $2^{-(\lambda+1)}\, n$.*

Proof. We prove the lemma by induction on λ. By Lemma 5, we know that $|Y_{\lambda^*}| \geq (\frac{1}{2} + \varepsilon)n$. Therefore, for $\lambda = 0$, the number of right vertices at augmentation distance greater than λ^* is at most $n/2$.

For some integer $\lambda \geq 0$, let the number of right vertices at augmentation distance greater than $\lambda + \lambda^*$ be at most $2^{-(\lambda+1)}\, n$. Consequently, there are at least $(1 + \varepsilon - 2^{-(\lambda+1)})n$ right vertices at augmentation distance no greater than $\lambda + \lambda^*$, i.e., $|Y_{\lambda+\lambda^*}| \geq (1 + \varepsilon - 2^{-(\lambda+1)})n$. By Lemma 8, the neighborhood of $Y_{\lambda+\lambda^*}$ includes at least $(1 - 2^{-(\lambda+2)})n$ left vertices. Both the mates of these left vertices and the free vertices are at augmentation distance at most $(\lambda + 1) + \lambda^*$. Hence, $|Y_{(\lambda+1)+\lambda^*}| \geq (1 + \varepsilon - 2^{-(\lambda+2)})n$, and no more than $2^{-(\lambda+2)}\, n$ right vertices are at augmentation distance greater than $(\lambda + 1) + \lambda^*$. □

Bounding the Size of the BFS Tree. Let v be a newly arrived left vertex, and let T_v be the random variable denoting the number of right vertices encountered before a shortest augmenting path starting at v is found (i.e., before the BFS reaches the first free vertex). Clearly, the insertion algorithm can add v to the current matching in $O(T_v)$ time and space. Then we use the assumption that the current matching and the sets Y_λ do not depend on the random choices of v, and we show that (i) T_v does not exceed $o(n)$ whp and (ii) the expectation of T_v does not exceed $2\, d^{\lambda^*+2}$.

We first assume that the bipartite graph B satisfies the conclusions of Lemmas 3 and 6 and contains an L-perfect matching which covers v. In addition, we assume that no rehash is carried out if there are too many right vertices in the BFS tree.

If at least one of the d neighbors of v is at augmentation distance at most λ, an augmenting path is found after at most $d^{\lambda+1}$ right vertices have been visited. Therefore, for every integer $\lambda \geq 0$, the probability that T_v exceeds $d^{\lambda+1}$ is at most $(1 - |Y_\lambda|/(1+\varepsilon)n)^d$.

By Lemma 8, there are at most $2^{-(\lambda+1)}\, n$ right vertices at augmentation distance greater than $\lambda + \lambda^*$. Hence, $1 - |Y_{\lambda^*+\lambda}|/(1+\varepsilon)n \leq 2^{-(\lambda+1)}$. Let β be a constant in the interval $(0, \frac{1}{2})$. Then

$$\mathbb{P}\mathrm{r}[T_v > d^{\lambda^*+1}\, n^{1-\beta}] < 2^{-(1-\beta)d \log_d n} = n^{-(1-\beta)d/\log d}.$$

Using $\beta = \frac{9}{10}$ and $d \geq 8$, we conclude that the probability that more than $d^{\lambda^*+1} n^{9/10}$ right vertices are encountered before an augmenting path is found does not exceed $n^{-24/10}$.

As for the expectation of T_v,

$$\begin{aligned}
\mathbb{E}[T_v] &= \sum_{t=1}^{\infty} \Pr[T_v \geq t] \\
&\leq d + \sum_{\lambda=0}^{\infty} d^{\lambda+2}\, \mathbb{P}\mathrm{r}[v \text{ has no neighbor in } Y_\lambda] \\
&\leq d + \sum_{\lambda=0}^{\infty} d^{\lambda+2} \left(1 - \frac{|Y_\lambda|}{(1+\varepsilon)n}\right)^d \\
&\leq d^{\lambda^*+2} + \frac{d^{\lambda^*+2}}{2^d} \sum_{\lambda=0}^{\infty} \left(\frac{d}{2^d}\right)^{\lambda},
\end{aligned}$$

where the last inequality follows from Lemma 8. For any $d \geq 8$, $d/2^d \leq \frac{1}{32}$ and the above sum can be bounded by $\frac{249}{248}\, d^{\lambda^*+2}$.

We also have to consider the contribution of several low probability events to the expectation of T_v. The probability that more than $d^{\lambda^*+1} n^{9/10}$ right vertices are encountered before an augmenting path is found is $O(n^{-24/10})$. In addition, for $d \geq 8$, the bipartite graph B violates the conclusions of Lemmas 3 or 6 with probability $O(n^{-4})$, and B does not contain an L-perfect matching with probability $O(n^{-12})$. Each of these events causes a rehash, whose cost is bounded by $O(n^2)$. Therefore, these low probability events have a negligible contribution to the expectation of T_v, which can be bounded by $2\, d^{\lambda^*+2}$.

We can choose $d = \Theta(\ln(1/\varepsilon))$. Using $\lambda^* = \Theta(\ln(1/\varepsilon))$, we conclude that the expectation of T_v is $(1/\varepsilon)^{O(\ln d)}$.

4. Experiments

Our theoretical analysis is not tight with respect to the constant factors and lower order terms in the relation between the worst case number of probes d and the waste of space εn. The analysis is even less accurate with respect to the insertion time. Since these quantities are important to judge how practical d-ary Cuckoo Hashing might be, we designed an experiment that can partially fill this gap. We decided to focus on a variant that looks promising in practice: We use d separate tables of size $(1+\varepsilon)n/d$ because then it is not necessary to re-evaluate the hash function that led to the old position of an element to be moved. Insertion uses a random walk, i.e., an element is randomly placed in of the d table positions available to it. If this position was previously occupied by another element, the displaced element randomly picks one of its $d-1$ remaining choices, etc., until a free table entry is found. The random walk insertion saves us some bookkeeping that would be needed for insertion by BFS. The source of the simulation program is avaiable at http://www.mpi-sb.mpg.de/ sanders/programs/cuckoo. Figure 2 shows the average number of probes needed for insertion as a function of the space utilization $1/(1+\varepsilon)$ for $d \in \{2, 3, 4, 5\}$. Since $1/\varepsilon$ is a lower bound, the y-axis is scaled by ε. We see that all schemes are close to the insertion time $1/\varepsilon$ for small utilization

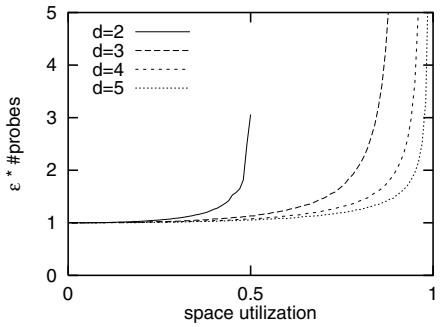

Fig. 2. Scaled average number of memory probes for insertion into a d-ary Cuckoo Hash table with 100 000 entries as a function of the memory utilization $n/10^5$ ($\varepsilon = 1 - n/10^5$). Starting from $n = 1000 \cdot k$ ($k \in \{1, \ldots, 100\}$), a random element is removed and a new random element is inserted. This is repeated 1000 times for each of 100 independent runs. Hash functions are full lookup tables filled with random elements generated using [18]. The curves stop when any insertion fails after 1000 probes.

and grow quickly as they approach a capacity threshold that depends on d. Increasing d strictly decreases average insertion time so that we get clear trade-off between worst case number of probes for accesses and average insertion time.

The maximum space utilization approaches one quickly as d is incremented. The observed thresholds were at 49 % for $d = 2$, 91 % at $d = 3$, 97 % at $d = 4$, and 99 % at $d = 5$.

5. Filter Hashing

In this section we describe and analyze *Filter Hashing*, a simple hashing scheme with worst case constant lookup time, that can be used in combination with essentially any other hashing scheme to improve the space efficiency of the latter. More precisely, Filter Hashing space efficiently stores almost all elements of a set. The remaining elements can then be stored using a less space efficient hashing scheme, e.g., [7].

To explain Filter Hashing, we again switch to the terminology of bipartite graphs. For a parameter γ, $0 < \gamma < 1$, we split the right vertices into $d = \Theta(\ln^2(1/\gamma))$ parts, called *layers*, of total size at most n. Each left vertex is associated with exactly one neighbor in each of the d layers, using hash functions as described below. A newly arrived vertex is always matched to an unmatched neighbor in the layer with the *smallest possible* number. The name filter hashing comes from the analogy of a particle (hash table element/left vertex) passing through a cascade of d filters (layers). If all the neighbors in the d layers have been matched, the vertex is not stored, i.e., it is left to the hashing scheme handling such "overflowing" vertices. We will show that this happens to at most γn elements whp.

If the hashing scheme used for the overflowing vertices uses linear space, a total space usage of $(1 + \varepsilon)\, n$ cells can be achieved for $\gamma = \Omega(\varepsilon)$. For example, if we use the dictionary of [7] to handle overflowing vertices, the space used for overflowing vertices is $O(\gamma n)$, and every insertion and lookup of an overflowing vertex takes constant time whp.

Even though this scheme exhibits relatively high constant factors in time and space, the effect on space and average time of the combined hashing scheme is small if we choose the constant γ to be small.

A hashing scheme similar to filter hashing, using $O(\log \log n)$ layers, was proposed in [2], but only analyzed for load factor less than $\frac{1}{2}$. Here, we use stronger tools and hash functions to get an analysis for space utilization arbitrarily close to 1.

What happens in the filtering scheme can be seen as letting the left vertices decide their mates using a multi-level balls and bins scenario, until the number of unmatched left vertices becomes small enough. The scheme gives a trade-off between the number of layers and the fraction γ of overflowing vertices.

We proceed to describe precisely the bipartite graph $B(L, R, E)$ used for the scheme, where $|L| = |R| = n$. We partition R into d layers R_i, $i = 1, \ldots, d$, where $d = \lceil \ln^2(4/\gamma) \rceil$ and $|R_i| = \lfloor (n/\ln(4/\gamma))/(1 - (1/\ln(4/\gamma)))^{i-1} \rfloor$. Suppose that $L \subseteq \{1, \ldots, m\}$ for some integer m, or, equivalently, that we have some way of mapping each vertex to a unique integer in $\{1, \ldots, m\}$. The edges connecting a vertex $v \in L$ to R_i, for $i = 1, \ldots, d$, are given by function values on v of the hash functions

$$h_i(x) = \left(\sum_{j=0}^{t} a_{ij} x^j \bmod p \right) \bmod |R_i|, \tag{4}$$

where $t = 12 \lceil \ln(4/\gamma) + 1 \rceil$, $p > mn$ is a prime number and the a_{ij} are randomly and independently chosen from $\{0, \ldots, p-1\}$.

For n larger than a suitable constant (depending on d), the total size $\sum_{i=1}^{d} |R_i|$ of the d layers is in the range

$$\left[\sum_{i=1}^{d} \frac{n}{\ln(4/\gamma)} \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{i-1} - d \; ; \; \sum_{i=1}^{\infty} \frac{n}{\ln(4/\gamma)} \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{i-1} \right]$$
$$= \left[n \left(1 - \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{d}\right) - d \; ; \; n \right] \subseteq \left[\left(1 - \frac{\gamma}{2}\right) n \; ; \; n \right].$$

From the description of filter hashing, it is straightforward that the worst case number of probes for inserting or accessing an element is $d = O(\ln^2(1/\varepsilon))$. Furthermore, if the evaluation of the hash functions is taken into account, the worst case insertion time and the worst case access time are $d\,t = O(\ln^3(1/\varepsilon))$. In the following we prove that at most γn left vertices overflow whp, and that the average time for a successful search is $O(\ln(1/\gamma))$. Both these results are implied by the following lemma.

Lemma 9. *For any constant γ, $0 < \gamma < 1$, for $d = \lceil \ln^2(4/\gamma) \rceil$ and n larger than a suitable constant, the number of left vertices matched to vertices in R_i is at least $(1 - \gamma/2)|R_i|$ for $i = 1, \ldots, d$ with probability $1 - O((1/\gamma)^{O(\log \log(1/\gamma))}(1/n))$.*

Proof. We use tools from [8] to prove that each of the layers has at least a fraction $(1 - \gamma/2)$ of its vertices matched to left vertices with probability 1–$O((1/\gamma)^{O(\log \log(1/\gamma))}(1/n))$. As there are $O(\ln^2(1/\gamma))$ layers, the probability that this happens for all layers is also $1 - O((1/\gamma)^{O(\log \log(1/\gamma))}(1/n))$.

The number of left vertices that are not matched to a vertex in layers $R_1, \ldots, R_{i-1}$ is at least $n_i = n - \sum_{j=1}^{i-1} |R_j|$. We have the inequalities

$$\left(1 - \frac{1}{\ln(4/\gamma)}\right)^{i-1} n \le n_i \le \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{i-1} n + i.$$

Consider the random variable $\mathsf{free}(R_{i+1})$ denoting the number of empty bins in the balls and bins scenario with n_{i+1} balls and $|R_{i+1}|$ bins, where the positions of the balls are given by a hash function of the form (4). This is a pessimistic estimate of the number of free vertices in layer $i+1$ of the hashing scheme, since n_{i+1} is a lower bound on the number vertices that are not matched to $R_1, \ldots, R_i$. We use tools for analyzing such a scenario from [8] to show that $\mathsf{free}(R_{i+1}) \le (\gamma/2)|R_{i+1}|$ with probability $1 - \mathrm{O}(1/n)$. Denote by b_j the number of balls in bin number j and let $C_k = \sum_{j=1}^{|R_{i+1}|} \binom{b_j}{k}$ be the number of "colliding k-sets of balls." Since $t/2$ is even, we have the following inequality [8, Proposition 6]:

$$\mathsf{free}(R_{i+1}) \le \sum_{k=0}^{t/2} (-1)^k C_k. \tag{5}$$

The "load factor" of the balls and bins scenario is $\alpha = n_{i+1}/|R_{i+1}| \ge \ln(4/\gamma)$. Since $p > mn$ we get from [8] that for $k \le t/2$,

$$\frac{(1 - \mathrm{O}(1/n_{i+1}))\, n_{i+1}\alpha^{k-1}}{k!} \le \mathbb{E}[C_k] \le \frac{(1 - \mathrm{O}(1/n_{i+1}))\, n_{i+1}\alpha^{k-1}}{k!}.$$

Thus we get the following upper bound:

$$\begin{aligned}&\mathbb{E}[\mathsf{free}(R_{i+1})] \\ &\quad \le \left(1 + \mathrm{O}\left(\frac{1}{n_{i+1}}\right)\right) n_{i+1} \sum_{k=0}^{t/2} \frac{(-1)^k \alpha^{k-1}}{k!} \le \frac{\frac{3}{2} n_{i+1} e^{-\alpha}}{\alpha} \le \frac{3\gamma}{8}|R_{i+1}|, \end{aligned} \tag{6}$$

where the second inequality uses $t \ge 10\alpha$ and that n (and thus n_{i+1}) is sufficiently large.

It is shown in [8] that $\mathrm{Var}(C_k) = \mathrm{O}(\alpha^{2k} n)$ for $k \le t/2$. Thus we can use Chebychev's inequality to bound the probability that $\mathsf{free}(R_{i+1})$ exceeds $(\gamma/2)|R_{i+1}|$:

$$\begin{aligned}\Pr[\mathsf{free}(R_{i+1}) > \frac{\gamma}{2}|R_{i+1}|] &\le \sum_{k=0}^{t/2} \Pr\left[\frac{|C_k - \mathbb{E}[C_k]| > (\gamma/8)|R_{i+1}|}{t/2+1}\right] \\ &\le \sum_{k=0}^{t/2} \frac{\mathrm{Var}(C_k)}{(\gamma/8|R_{i+1}|/(t/2+1))^2} \\ &= O\left(\left(\frac{1}{\gamma}\right)^{\mathrm{O}(\log\log(1/\gamma))} \frac{1}{n}\right). \end{aligned}$$

□

Lemma 9 implies that there are at most $(\gamma/2)n$ of the n right side vertices that are not part of $R_1, \ldots, R_d$, and with probability $1 - \mathrm{O}(1/n)$ there are at most $(\gamma/2)n$ vertices

in the layers that are not matched. Thus, with probability $1 - O(1/n)$ no more than γn vertices overflow.

The expected average time for a successful search can be bounded as follows. The number of elements with search time $i \leq d$ is at most $|R_i|$, and the probability that a random left vertex overflows is at most $\gamma + O(1/n)$, i.e., the expected total search time for all elements is bounded by

$$\begin{aligned}
&\left(\gamma + O\left(\frac{1}{n}\right)\right) nd + \sum_{i=1}^{d} |R_i| i \\
&\quad \leq \left(\gamma + O\left(\frac{1}{n}\right)\right) \left\lceil \ln^2\left(\frac{4}{\gamma}\right) \right\rceil n + \frac{n}{\ln(4/\gamma)} \sum_{i=0}^{\infty} \left(1 - \frac{1}{\ln(4/\gamma)}\right)^i i \\
&\quad = O\left(n \ln\left(\frac{4}{\gamma}\right)\right).
\end{aligned}$$

The expected time to perform a rehash in case too many elements overflow is $O(\ln(1/\gamma)n)$. Since the probability that this happens for any particular insertion is $O((1/\gamma)^{O(\log\log(1/\gamma))}(1/n))$, the expected cost of rehashing for each insertion is $(1/\gamma)^{O(\log\log(1/\gamma))}$. Rehashes caused by the total number of elements (including those marked deleted) exceeding n have a cost of $O(\ln(1/\gamma)/\gamma)$ per insertion and deletion, which is negligible.

6. Conclusions and Open Problems

From a practical point of view, d-ary Cuckoo Hashing seems a very advantageous approach to space efficient hash tables with worst case constant access time. Both worst case access time and average insertion time are very good. It also seems that one could make *average* access time quite small. A wide spectrum of algorithms could be tried out from maintaining an optimal placement of elements (via minimum weight bipartite matching) to simple and fast heuristics.

Theoretically, there are two main open questions. The first concerns tight (high probability) bounds for the insertion time. The second question is whether the analysis of d-ary Cuckoo Hashing also works for practical, constant time evaluable hash functions. Dietzfelbinger has suggested [6] the following interesting solution: A very simple hash function based on polynomials of constant degree is used to partition the elements into $\log n$ disjoint groups of size at most $(1 + \varepsilon/2)n/\log n$ [15]. Now space linear in the size of one group ($O(dn/\log n)$) is invested to obtain an emulation of d uniform hash functions within one group. This can be achieved using recent constructions by Östlin and Pagh [21], or Dietzfelbinger and Wölfel [10]. Each group is stored in a table with $(1 + \varepsilon)n/\log n$ entries. The main trick is that the same d hash functions can be used for *all* the groups so that the total space needed for the hash functions remains sublinear.

Filter Hashing is inferior in practice to d-ary Cuckoo Hashing but it might have specialized applications. For example, it could be used as a *lossy* hash table with worst case constant *insertion* time. This might make sense in real time applications

where delays are not acceptable whereas losing some entries might be tolerable, e.g., for gathering statistic information on the system. In this context, it would be theoretically and practically interesting to give performance guarantees for simpler hash functions.

References

[1] R. P. Brent. Reducing the retrieval time of scatter storage techniques. *Communications of the ACM*, 16(2):105–109, 1973.

[2] A. Z. Broder and A. R. Karlin. Multilevel adaptive hashing. In *Proceedings of the* 1*st Annual ACM–SIAM Symposium on Discrete Algorithms* (*SODA* '90), pages 43–53. ACM Press, New York, 2000.

[3] A. Brodnik and J. I. Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.

[4] J. G. Cleary. Compact hash tables using bidirectional linear probing. *IEEE Transactions on Computers*, C-33(9):828–834, September 1984.

[5] R. Diestel. *Graph Theory*, 2nd edition. Springer-Verlag, New York, 2002.

[6] M. Dietzfelbinger. Personal communication, 2003.

[7] M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger. Polynomial hash functions are reliable (extended abstract). In *Proceedings of the* 19*th International Colloquium on Automata*, *Languages and Programming* (*ICALP* '92), pages 235–246. Volume 623 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.

[8] M. Dietzfelbinger and T. Hagerup. Simple minimal perfect hashing in less space. In *Proceedings of the* 9*th European Symposium on Algorithms* (*ESA* '01), pages 109–120. Volume 2161 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2001.

[9] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

[10] M. Dietzfelbinger and P. Wölfel. Almost random graphs with simple hash functions. In *Proceedings of the* 35*th Annual ACM Symposium on Theory of Computing* (*STOC* '03), 2003.

[11] D. P. Dubhashi and D. Ranjan. Balls and Bins: a study in negative dependence. *Random Structures & Algorithms*, 13:99–124, 1998.

[12] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[13] G. H. Gonnet and J. I. Munro. Efficient ordering of hash tables. *SIAM Journal on Computing*, 8(3):463–478, 1979.

[14] J. E. Hopcroft and R. M. Karp. An $O(n^{5/2})$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[15] C. P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *Theoretical Computer Science*, 71(1):95–132, 1990.

[16] J. A. T. Maddison. Fast lookup in hash tables with direct rehashing. *The Computer Journal*, 23(2):188–189, May 1980.

[17] E. G. Mallach. Scatter storage techniques: a uniform viewpoint and a method for reducing retrieval times. *The Computer Journal*, 20(2):137–140, May 1977.

[18] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998. http://www.math.keio.ac.jp/ matumoto/emt.html.

[19] R. Motwani. Average-case analysis of algorithms for matchings and related problems. *Journal of the ACM*, 41(6):1329–1356, November 1994.

[20] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, 1995.

[21] A. Östlin and R. Pagh. Uniform hashing in constant time and linear space. In *Proceedings of the* 35*th Annual ACM Symposium on Theory of Computing* (*STOC* '03), 2003.

[22] R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, 31(2):353–363, 2001.

[23] R. Pagh and F. F. Rodler. Cuckoo hashing. In *Proceedings of the* 9*th European Symposium on Algorithms* (*ESA* '01), pages 121–133. Volume 2161 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2001.
[24] R. Raman and S. Srinivasa Rao. Succinct dynamic dictionaries and trees. In *Proceedings of the* 30*th International Colloquium on Automata Languages and Programming*, pages 345–356. Volume 2719 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2003.
[25] R. L. Rivest. Optimal arrangement of keys in a hash table. *Journal of the ACM*, 25(2):200–209, 1978.
[26] P. Sanders. Asynchronous scheduling of redundant disk arrays. In *Proceedings of the* 12*th ACM Symposium on Parallel Algorithms and Architectures*, pages 89–98, 2000.
[27] P. Sanders. Reconciling simplicity and realism in parallel disk models. In *Proceedings of the* 12*th ACM–SIAM Symposium on Discrete Algorithms*, pages 67–76, Washington, DC, 2001.
[28] P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *Proceedings of the* 11*th ACM–SIAM Symposium on Discrete Algorithms*, pages 849–858, 2000.
[29] A. C.-C. Yao. Uniform hashing is optimal. *Journal of the ACM*, 32(3):687–693, 1985.

Received May 15, 2003, *and in revised form November* 24, 2003, *and in final form March* 23, 2004.
Online publication December 13, 2004.