Space Efficient Hash Tables With Worst Case Constant Access Time*

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

[1] Max Planck Institute for Computer Science, Saarbrücken, Germany. [fotakis,sanders]@mpi-sb.mpg.de
[2] IT University of Copenhagen, Denmark. pagh@itu.dk
[3] Computer Technology Institute (CTI), Greece. spirakis@cti.gr

Abstract. We generalize Cuckoo Hashing [19] to *d-ary Cuckoo Hashing* and show how this yields a simple hash table data structure that stores n elements in $(1+\epsilon)\,n$ memory cells, for any constant $\epsilon > 0$. Assuming uniform hashing, accessing or deleting table entries takes at most $d = \mathrm{O}(\ln \frac{1}{\epsilon})$ 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 $(1+\epsilon)\,n$ space, and supports satellite information. Experiments indicate that $d = 4$ choices suffice for $\epsilon \approx 0.03$. We also describe a hash table data structure using explicit constant time hash functions, using at most $d = \mathrm{O}(\ln^2 \frac{1}{\epsilon})$ probes in the worst case.

A corollary is an expected linear time algorithm for finding maximum cardinality matchings in a rather natural model of sparse random bipartite graphs.

1 Introduction

The efficiency of many programs crucially depends on hash table data structures which support constant expected access time for any type of elements that allow to compute hash functions in constant time. We also know hash table data structures that support *worst case* constant access time for quite some time [10, 7]. 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* [19]: Each element is mapped to two tables t_1 and t_2 of size $(1+\gamma)\,n$ using two hash functions h_1 and h_2, for any $\gamma > 0$. A factor above two in space expansion is sufficient to ensure with high probability (henceforth "whp."[1])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 entries 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 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 $(1+\epsilon)\,n$. 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 $(1+\epsilon)\,n$. The access time is $\mathrm{O}(\ln \frac{1}{\epsilon})$ which is in some sense optimal, and the expected insertion time is also constant. The proposed

* 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 this author was at BRICS, Aarhus University, Denmark.

*** Part of this work was done while the author was at MPII.

[1] In this paper "whp." will mean "with probability $1 - O(1/n)$".

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. 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 [19] breaks down for $d \geq 3$. Therefore we develop new approaches. Section 2 constitutes the main part of the paper and gives an analysis of the simple algorithm outlined above for the case that hash functions are truly random and that no element is deleted and later inserted again.[2]

Section 3 complements this analysis by experiments that indicate that Cuckoo Hashing is even better in practical situations. 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 the lower bound of 10 probes.

In Section 4 we present *Filter Hashing*, an alternative to d-ary Cuckoo Hashing that uses *explicit* constant time evaluable hash functions (polynomial hash functions of degree $\mathrm{O}(\ln \frac{1}{\epsilon})$). It has the same performance as d-ary Cuckoo Hashing except that it uses $d = \mathrm{O}(\ln^2 \frac{1}{\epsilon})$ probes for an access in the worst case.

A novel feature of both d-ary Cuckoo Hashing and Filter Hashing is that we use hash tables having size only a fraction of the number of elements hashed to them. This means that high space utilization is ensured, even though there is only one possible location for an element in each 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 $B = n \log(em/n) - \Theta(n^2/m) - O(\log n)$. This is roughly $n \log n$ bits less than, say, a sorted list of elements. If $\log m$ is large compared to $\log n$, 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 $(1+\epsilon)B + O(n)$ bits, under the assumption that a truly random permutation on the key space is available. The expected average time for lookups and insertions is $O(1/\epsilon^2)$, as in ordinary linear probing. A space usage of $B + o(n) + O(\log \log m)$ bits was obtained in [18] 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 $O(B)$ in a dictionary that has worst case constant lookup time and

[2] For theoretical purposes, the 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.

amortized expected constant time for insertions and deletions. The space usage was recently improved to $B+o(B)$ bits by Raman and Rao [20]. 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 $(1+\epsilon)\,n$ 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 $(1+\epsilon)\,B+O(n)$ bits using *quotienting* (as in [4, 18]). 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 $O(\ln(1/\epsilon))$ bound on lookup time matches the *average* case bound of classical open addressing schemes like double hashing. Yao [26] 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 have been proposed that improve or remove the dependence on ϵ of the average lookup time by moving around elements [1, 11, 13, 14, 21].

The *worst case* retrieval cost of the classical open addressing schemes is $\Omega(\log n)$. Bounding the worst case retrieval cost in open addressing schemes was investigated by Rivest [21], 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 $\epsilon \leq 1$).

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 [24, 23] 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 [23] this relation was only tabulated for small values of n/D.

The analysis of insertion time uses expansion properties of random bipartite graphs. Motwani [17] uses expansion properties to show that the algorithm by Hopcroft and Karp [12] finds perfect matchings in random bipartite graphs with $m > n \ln n$ edges in time $\mathrm{O}(m \log n / \log\log n)$. He shows an $\mathrm{O}(m \log n / \log d)$ bound for the *d-out* model of random bipartite graphs, where all nodes are constrained to have degree at least $d > 4$. Our analysis of insertion can be understood as an analysis of a simple incremental path algorithm for finding maximum perfect matchings in a random bipartite graph where n nodes on the left side are constrained to have constant degree d whereas there are $(1+\epsilon)\,n$ 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 where it is unrealistic to assume a lower bound on the degree of a right node (e.g., [24, 22, 23]). Under these conditions we get *linear* run time even for very sparse graphs using a very simple algorithm that has the additional advantage to allow 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 [17] is that our graphs have weaker expansion properties because they are less dense (or less regular for the d-out model).

2 d-ary Cuckoo Hashing

A natural way to define and analyze d-ary Cuckoo Hashing is through matchings in asymmetric bipartite graphs. In particular, the elements can be thought of as the left vertices and the memory cells can be thought of as the right vertices of a bipartite graph $B(L, R, E)$. For d-ary Cuckoo Hashing, the number of right vertices is $(1+\epsilon)$ times the number of left vertices. An edge connecting a left vertex to a right vertex indicates that the corresponding element can be stored at the corresponding memory cell. For d-ary Cuckoo Hashing, the edge set of the bipartite graph B is a random set determined by letting each left vertex select exactly d neighbors randomly and independently (with replacement) from the set of right vertices. Any one-to-one assignment of elements/left vertices to memory cells/right vertices forms a matching in B. Since every element is stored at some cell, this matching is L-perfect, i.e. it covers all the left vertices.

In the following, we only consider bipartite graphs resulting from d-ary Cuckoo Hashing, i.e. graphs $B(L, R, E)$ where $|L| = n$, $|R| = (1+\epsilon)\,n$, and each left vertex has exactly d neighbors selected randomly and independently (with replacement) from R. We also assume that the left vertices arrive (along with their d random choices/edges) one-by-one in an arbitrary order and the insertion algorithm incrementally maintains an L-perfect matching in B.

Having fixed an L-perfect matching M, we can think of B as a directed graph, where all the edges of E are directed from left to right, except for the edges of M, which are directed from right to left. Hence, the matching M simply consists of the edges directed from right to left. In addition, the set of *free vertices* $F \subseteq R$ simply consists of the right vertices with no outgoing edges.

When a new left vertex v arrives, all its edges are considered as outgoing (i.e. directed from left to right), since v is not currently matched. Then, any directed path from v to a F is an augmenting path for M, because if we reverse the directions of all the edges along such a path, we obtain a new matching M' which also covers v. The insertion algorithm we analyze always augments M along a shortest directed path from v to a vertex in F. Such a path can be found by a Breadth First Search (BFS) starting from v in the directed version of B.

2.1 Existence of an L-Perfect Matching

We start by showing that for appropriately large values of d, d-ary Cuckoo Hashing leads to bipartite graphs that contain an L-perfect matching whp.

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

Proof Sketch. By Hall's Theorem (e.g. [5], Chapter 2), it suffices to show that for any integer κ, $2 \leq \kappa \leq n$, every subset of left vertices of size κ has at least κ neighbors in R with probability at least $1 - \mathrm{O}(n^{4-2d})$. The whole proof can be found in the appendix. □

By applying our analysis for particular values of ϵ, we obtain that if $\epsilon \geq 0.57$ and $d = 3$, if $\epsilon \geq 0.19$ and $d = 4$, and if $\epsilon \geq 0.078$ and $d = 5$, B contains an L-perfect matching whp. The experiments in Section 3 indicate that even smaller values of ϵ are possible.

In addition, we can show that this bound for d is best possible within a small constant multiplicative factor.

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

Proof Sketch. In the appendix we show that if $d < (1 + \epsilon)\ln(1/\epsilon)$, there are more than ϵn isolated right vertices whp. □

2.2 The Average Insertion Time of d-ary Cuckoo Hashing

As stated in the introduction, we can restrict our attention to the case where a left vertex that has been deleted from the hash table cannot be reinserted. This means that we avoid any dependencies among the random edges/choices of a newly arrived left vertex and the current matching. In addition, we assume that the bipartite graph contains an L-perfect matching, which happens whp. according to Lemma 1.

Theorem 1. *For any positive $\epsilon \leq 1/5$ and $d \geq 5 + 3\ln(1/\epsilon)$, the BFS-based incremental algorithm that augments along a shortest augmenting path needs $(1/\epsilon)^{O(\log d)}$ expected time per left vertex/element to maintain an L-perfect matching in B.*

The proof of Theorem 1 consists of three parts. We first prove that within some set consisting of almost half of the right vertices, the number of vertices having a directed path to the set of free vertices F of length at most 2λ grows exponentially with λ, whp. We call this the *expansion property.* We next prove that for the remaining right vertices, the number of right vertices having no path to F of length at most 2λ 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 appropriate choices of d, d-ary Cuckoo Hashing results in bipartite graphs that are good expanders, whp. Finally, we put the expansion property and the shrinking property together to show that the expected insertion time per element is constant.

Remark. Using the same techniques, we can prove that Theorem 1 holds for any given constant $\epsilon \in (0, 1)$ by appropriately restricting the range of the constant δ. However, for ease of exposition, we restrict our attention to the most interesting case of small ϵ.

Proof of Theorem 1. In the proof of Theorem 1, we are interested in bounding the distance (respecting the edge directions) of matched right vertices from the set of free vertices F, because a shortest directed path from a matched right vertex u to F can be used for the insertion of a newly arrived left vertex which has an edge incident to u.

We measure the distance of a vertex v from 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 sometimes refer to this distance as the *augmentation distance* of v. Notice that this distance depends on the current matching M.

We can define the augmentation distance for both the right and the left vertices inductively starting from F. Any free right vertex is at augmentation distance 0. The left vertices having a (free) edge incident to F are at augmentation distance 1. The mate of a left vertex being at augmentation distance i is also at augmentation distance i. A left vertex v is at augmentation distance $i+1$ if v has a free edge incident to a right vertex being at augmentation distance i and no neighbor of v (reached by a free edge) has augmentation distance less than i.

The augmentation distance of a vertex v essentially measures the depth at which a Breadth First Search starting from v reaches a free vertex for the first time. Therefore, the complexity of searching for a shortest directed path from v to F can be bounded using the augmentation distance of v.

Throughout the proof of Theorem 1, we use X to denote subsets of left vertices and Y to denote subsets of right vertices. Given a set $X \subseteq L$, we use $M(X)$ to denote the set of X's mates with respect to the current matching. Given a set of vertices S, we use $\Gamma(S)$ to denote the set of S's neighbors, i.e. $\Gamma(S) = \{v : \{u, v\} \in E, u \in S\}$. Notice that the definition of $\Gamma(S)$ does not refer to any matching. Consequently, any time we use $\Gamma(S)$, we consider the edges as undirected.

For any integer λ, we use Y_λ to denote the set of right vertices at augmentation distance at most λ, and X_λ to 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 $\lambda = 0$, we set $Y_0 = F$ and $X_0 = \emptyset$. For any integer $\lambda \geq 1$, we have $X_{\lambda+1} = \Gamma(Y_\lambda)$ and $Y_{\lambda+1} = M(X_{\lambda+1}) \cup F$.

The Expansion Property. We first prove that if d is chosen appropriately large, for any small constant δ, any set of right vertices Y that is not too close to size $n/2$ has at least $(1+\delta)|Y|$ neighbors in L whp. (cf. Lemma 3). This implies that the number of right vertices at augmentation distance at most λ is at least $\epsilon(1 + (1+\delta)^\lambda)n$, as long as λ is so small that this number does not exceed $n/2$ (cf. Lemma 4).

Lemma 3. *Given a constant $\epsilon \in (0, 1/4)$, let δ be any positive constant not exceeding $\frac{4(1-4\epsilon)}{1+4\epsilon}$, and let d be any integer such that $d \geq 3 + 2\delta + 2\epsilon(1+\delta) + (2+\delta)\epsilon \ln\left(\frac{1+\epsilon}{\epsilon}\right) / \ln(1+\epsilon)$. Then, any set of right vertices $Y \subseteq R$ of cardinality $\epsilon n \leq |Y| \leq \frac{n}{2(1+\delta)}$ has at least $(1+\delta)|Y|$ neighbors in L with probability $1 - 2^{-\Omega(n)}$.*

Proof Sketch. In the appendix we show that any subset of left vertices X, $\frac{n}{2} \le |X| \le (1-(1+\delta)\epsilon)n$, has at least $(1+\epsilon)n - \frac{n-|X|}{1+\delta}$ neighbors whp., and that any bipartite graph B having this property also satisfies the conclusion of the lemma. $\square$

The following lemma concludes the proof of the expansion property.

Lemma 4. *Given a constant $\epsilon \in (0, 1/4)$, let δ be any positive constant not exceeding $\frac{4(1-4\epsilon)}{1+4\epsilon}$, and let $B(L, R, E)$ be any bipartite graph satisfying the conclusion of Lemma 3. Then, for any integer λ, $1 \le \lambda \le \log_{(1+\delta)}\left(\frac{1}{2\epsilon}\right)$, the number of right vertices at augmentation distance at most λ is at least $\epsilon(1+(1+\delta)^\lambda)n$.*

Proof Sketch. The proof is by induction on λ. If the lemma holds for some λ, the conclusion of Lemma 3 implies that it also holds for $\lambda+1$. The whole proof is given in the appendix. $\square$

For $\epsilon \le 1/5$ we can take $\delta = 1/3$. Then, using the fact that $\ln(1+\epsilon) > \epsilon - \epsilon^2/2$, the requirement of Lemma 3 on d can be seen to be satisfied if $d \ge 5 + 3\ln(1/\epsilon)$.

The Shrinking Property. By the expansion property, there is a set consisting of nearly half of the right vertices that have augmentation distance smaller than some number $\lambda^\star$. The second thing we show is that for any constant $\gamma > 0$, any set of left vertices X, $|X| \le \frac{n}{2(1+\gamma)}$, has at least $(1+\gamma)|X|$ neighbors in R whp. (cf. Lemma 9). This implies that the number of right vertices being at augmentation distance greater than $\lambda^\star + \lambda$ decreases exponentially with λ (cf. Lemma 6).

Lemma 5. *Let γ be any positive constant, and let $d \ge (1+\log e)(2+\gamma) + \log(1+\gamma)$ be an integer. Then, any set of right vertices Y, $|Y| \ge (1/2+\epsilon)n$, has at least $n - \frac{(1+\epsilon)n-|Y|}{(1+\gamma)}$ neighbors, whp.*

Proof Sketch. In the appendix we show that any set of left vertices X, $|X| \le \frac{n}{2(1+\gamma)}$, has at least $(1+\gamma)|X|$ neighbors whp. and that any bipartite graph B having this property also satisfies the conclusion of the lemma. $\square$

By applying the proof of Lemma 5 for $\gamma = 0.55$, we obtain the following corollary.

Corollary 1. *For $d \ge 6$, any subset of right vertices Y, $|Y| \ge (1/2+\epsilon)n$, has at least $n - \frac{(1+\epsilon)n-|Y|}{1.55}$ neighbors with probability at least $1 - \mathrm{O}(n^{-2})$.*

The following lemma concludes the proof of the shrinking property. Its proof is similar to the proof of Lemma 4 and is given in the appendix.

Lemma 6. *Given a constant $\epsilon \in (0, 1/4)$, let δ be any positive constant not exceeding $\frac{4(1-4\epsilon)}{1+4\epsilon}$ and let γ be any positive constant. In addition, let $B(L, R, E)$ be any bipartite graph satisfying the conclusions of both Lemma 3 and Lemma 5 and let $\lambda^\star = \left\lceil \log_{(1+\delta)}\left(\frac{1}{2\epsilon}\right)\right\rceil$. Then, for any integer $\lambda \ge 0$, the number of right vertices at augmentation distance greater than $\lambda + \lambda^\star$ is at most $\frac{n}{2(1+\gamma)^\lambda}$.*

Bounding the Average Insertion Time. We can now put everything together to bound the average insertion time of d-ary Cuckoo Hashing, thus concluding the proof of Theorem 1.

Let T_v be the random variable denoting the time required by the insertion algorithm to add a newly arrived left vertex v to the current matching. We will now use the assumption that the current matching and the sets Y_λ do not depend on the random choices of v.

At first we assume that the bipartite graph B satisfies the conclusions of both Lemma 3 and Lemma 5, and we use Lemmas 4 and 6 to bound the expectation of T_v.

If at least one of the d neighbors of v is at augmentation distance at most λ, the BFS will find an augmenting path starting at v in time $\mathrm{O}(d^{\lambda+1})$. Therefore, for any integer $\lambda \geq 0$, with probability at least $1 - \left(1 - \frac{|Y_\lambda|}{(1+\epsilon)n}\right)^d$, T_v is $\mathrm{O}(d^{\lambda+1})$. Hence, the expectation of T_v can be bounded by

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

If the graph B satisfies the conclusions of both Lemma 3 and Lemma 5, we can bound the expectation of T_v by

$$\mathbb{E}[T_v] \leq \mathrm{O}(d^{\lambda^\star+2}) + \frac{d^{\lambda^\star+2}}{2^d} \sum_{\lambda=0}^{\infty} \left(\frac{d}{(1+\gamma)^d}\right)^\lambda .$$

If we choose $\gamma = 0.55$ and $d \geq 6$ (cf. Corollary 1), the quantity $\frac{d}{(1+\gamma)^d}$ becomes smaller than $1/2$. Therefore, the expectation of T_v can be bounded by $d^{\mathrm{O}(\lambda^\star)}$.

On the other hand, with probability $\mathrm{O}(n^{-2})$ the bipartite graph B does not have the desired properties (i.e. it violates the conclusions of Lemma 3 or Lemma 5). However, T_v is always bounded by $\mathrm{O}(n)$ and therefore this low probability event has a negligible contribution to the expectation of T_v. □

3 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 $(1 + \epsilon)/d$ because then it is not necessary to reevaluate the hash function that led to the old position of an element to be moved. Insertion uses a random walk, i.e., an element to be allocated randomly picks one of its d choices even

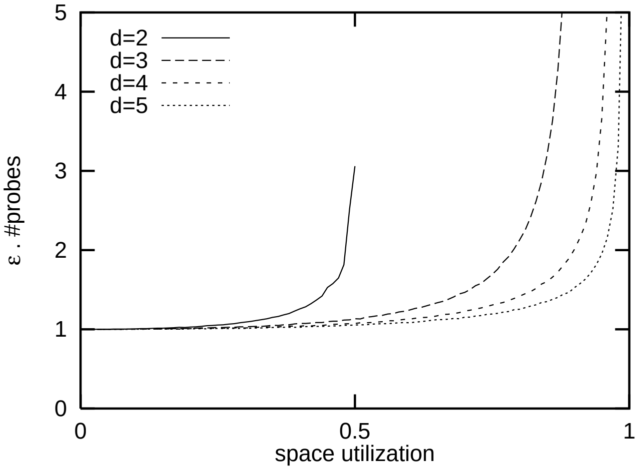

Fig. 1. Scaled number of memory probes for d-ary Cuckoo Hashing as a function of the memory utilization $1/(1+\epsilon)$. The results are the average of 100 independent runs on a table with 100 000 entries. In each run, for $n = 1000 \cdot k$, $k \in \{1, \ldots, 100\}$ 1000 random elements are removed and a new element is inserted immediately. The hash functions are full lookup tables filled with random elements generated using the Mersenne Twister random number generator [15]. The curves stop when any insertion fails after 1000 probes.

if the space is occupied. In the latter case, 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. Fig. 1 shows the average number of probes needed for insertion as a function of the space utilization $1/(1+\epsilon)$ for $d \in \{2, 3, 4, 5\}$. Since $1/\epsilon$ is a lower bound, the y-axis is scaled by ϵ. We see that all schemes are close to the insertion time $1/\epsilon$ for small utilization and grow quickly as they approach a capacity threshold that depends on d. Increasing d strictly decreases expected insertion time so that we get clear trade-off between worst case access time guarantees 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 $d = 5$.

4 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., [6].

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 $(1+\epsilon)\,n$ cells can be achieved for $\gamma = \Omega(\epsilon)$. For example, if we use the dictionary of [6] 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 $\mathrm{O}(\log\log n)$ layers, was proposed in [2], but only analyzed for load factor less than 1/2. Here, we use stronger tools and hash functions to get an analysis for load factors 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 $|R_i| = \left\lfloor \frac{n}{\ln(4/\gamma)} \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{i-1} \right\rfloor$. 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

$$h_i(x) = (\sum_{j=0}^{t} a_{ij} x^j \bmod p) \bmod |R_i| \tag{1}$$

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

$$\begin{aligned} &\left[\sum_{i=1}^{d} \frac{n}{\ln(4/\gamma)} \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{i-1} - d\ ;\ \sum_{i=1}^{\infty} \frac{n}{\ln(4/\gamma)} \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{i-1}\right] \\ &= \left[n\left(1 - \left(1 - \frac{1}{\ln(4/\gamma)}\right)^{d}\right) - d\ ;\ n\right] \subseteq \left[(1 - \tfrac{\gamma}{2})\,n\ ;\ n\right] \end{aligned}$$

From the description of filter hashing, it is straightforward that the worst case insertion time and the worst case access time are at most d. In the following, we prove that at most γn left vertices overflow whp., and that the average time for a successful search is $\mathrm{O}(\ln(1/\gamma))$. Both these results are implied by the following lemma.

Lemma 7. *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 $i = 0, \ldots, d$ with probability $1 - O\left(\left(\frac{1}{\gamma}\right)^{O(\log\log(\frac{1}{\gamma}))} \frac{1}{n}\right)$.*

Proof. In the appendix we prove that each of the layers has at least a fraction $(1 - \gamma/2)$ of its vertices matched to left vertices with probability $1 - \mathrm{O}(\frac{1}{n})$. As there are $O(\ln^2(1/\gamma))$ layers, the probability that this happens for all layers is also $1 - O((\frac{1}{\gamma})^{O(\log\log(\frac{1}{\gamma}))}\frac{1}{n})$. $\square$

To conclude, there are at most $\frac{\gamma}{2}n$ of the n right side vertices that are not part of $R_1, \ldots, R_d$, and with probability $1 - O(\frac{1}{n})$ there are at most $\frac{\gamma}{2}n$ vertices in the layers that are not matched. Thus, with probability $1 - O(\frac{1}{n})$ 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 $i \le d$ is at most $|R_i|$, and the probability that a random left vertex overflows is at most $\gamma + O(\frac{1}{n})$, i.e., the expected total search time for all elements is bounded by:

$$(\gamma + O(\tfrac{1}{n}))\, nd + \sum_{i=1}^{d} |R_i| i \le (\gamma + O(\tfrac{1}{n}))\lceil \ln^2(4/\gamma) \rceil\ n + \frac{n}{\ln(4/\gamma)} \sum_{i=0}^{\infty} \left(1 - \frac{1}{\ln(4/\gamma)}\right)^i i$$
$$= O(n \ln(1/\gamma))\ .$$

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 $O((\frac{1}{\gamma})^{O(\log\log(\frac{1}{\gamma}))}\frac{1}{n})$, the expected cost of rehashing for each insertion is $(\frac{1}{\gamma})^{O(\log\log(\frac{1}{\gamma}))}$. Rehashes causes by the total number of elements (including those marked deleted) exceeding n have a cost of $O(\ln(\frac{1}{\gamma})/\gamma)$ per insertion and deletion, which is negligible.

5 Conclusions and Open Problems

From a practical point of view, d-ary Cuckoo Hashing seems an almost ideal 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, many open questions remain. Can we work with (practical) hash functions that can be evaluated in constant time? What are tight (high probability) bounds for the insertion time (for an arbitrary mix of insertions and deletions)?

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 loosing 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 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 43–53. ACM Press, 2000.
3. A. Brodnik and J. I. Munro. Membership in constant time and almost-minimum space. *SIAM J. Comput.*, 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.* Springer, 2nd edition, 2002.
6. M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger. Polynomial hash functions are reliable (extended abstract). In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, volume 623 of *Lecture Notes in Computer Science*, pages 235–246. Springer-Verlag, 1992.
7. 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 J. Comput.*, 23(4):738–761, 1994.
8. M. Dietzfelbinger and T. Hagerup. Simple minimal perfect hashing in less space. In *Proceedings of the 9th European Symposium on Algorithms (ESA '01)*, volume 2161 of *Lecture Notes in Computer Science*, pages 109–120. Springer-Verlag, 2001.
9. D. P. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *RSA: Random Structures & Algorithms*, 13:99–124, 1998.
10. M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. Assoc. Comput. Mach.*, 31(3):538–544, 1984.
11. G. H. Gonnet and J. I. Munro. Efficient ordering of hash tables. *SIAM J. Comput.*, 8(3):463–478, 1979.
12. J. E. Hopcroft and R. M. Karp. An $\mathrm{O}(n^{5/2})$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.
13. J. A. T. Maddison. Fast lookup in hash tables with direct rehashing. *The Computer Journal*, 23(2):188–189, May 1980.
14. 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.
15. M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACMTMCS: ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998. `http://www.math.keio.ac.jp/~matumoto/emt.html`.
16. J. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
17. R. Motwani. Average-case analysis of algorithms for matchings and related problems. *Journal of the ACM*, 41(6):1329–1356, November 1994.
18. R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput.*, 31(2):353–363, 2001.
19. Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In *Proceedings of the 9th European Symposium on Algorithms (ESA '01)*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer-Verlag, 2001.
20. R. Raman and S. Srinivasa Rao. Dynamic dictionaries and trees in near-minimum space. Manuscript, 2002.
21. R. L. Rivest. Optimal arrangement of keys in a hash table. *J. Assoc. Comput. Mach.*, 25(2):200–209, 1978.
22. P. Sanders. Asynchronous scheduling of redundant disk arrays. In *12th ACM Symposium on Parallel Algorithms and Architectures*, pages 89–98, 2000.
23. P. Sanders. Reconciling simplicity and realism in parallel disk models. In *12th ACM-SIAM Symposium on Discrete Algorithms*, pages 67–76, Washington DC, 2001.
24. P. Sanders, S. Egner, and J. Korst. Fast concurrent access to parallel disks. In *11th ACM-SIAM Symposium on Discrete Algorithms*, pages 849–858, 2000.
25. T. Worsch. Lower and upper bounds for (sums of) binomial coefficients. Technical Report IB 31/94, Universität Karlsruhe, 1994.
26. A. C.-C. Yao. Uniform hashing is optimal. *J. Assoc. Comput. Mach.*, 32(3):687–693, 1985.

A Proof Details

A.1 Preliminaries

In the proofs of Lemmas 1, 8 and 9, we use the following upper bound for binomial coefficients (e.g. [25]):

Proposition 1. *For any positive integer n and any $\mu \in (0,1]$,*

$$\binom{n}{\mu n} \leq \left(\left(\frac{1}{1-\mu}\right)^{1-\mu}\left(\frac{1}{\mu}\right)^{\mu}\right)^{n}.$$

A.2 Proof of Lemma 1

By Hall's Theorem (e.g. [5], Chapter 2), it suffices to show that any set $X \subseteq L$, $|X| \geq 2$, has at least $|X|$ neighbors in R whp, where the neighborhood of a vertex is taken without considering any edge directions.

For some integer κ, $2 \leq \kappa \leq n$, let P_κ be the probability that there is a set $X \subseteq L$ of cardinality κ which has less than κ neighbors. The probability that B does not have a matching of size n is bounded by the sum of the probabilities P_κ for all $\kappa \in \{2, \ldots, n\}$.

After having fixed κ, we fix a pair of sets $X \subseteq L$ and $Y \subseteq R$ both of size κ. The probability that $\Gamma(X) \subseteq Y$ is

$$\left(\frac{\kappa}{(1+\epsilon)n}\right)^{d\kappa}.$$

Since the sets X and Y can be chosen in $\binom{n}{\kappa}$ and $\binom{(1+\epsilon)n}{\kappa}$ different ways respectively, the probability P_κ cannot exceed the following quantity:

$$\binom{n}{\kappa}\binom{(1+\epsilon)n}{\kappa}\left(\frac{\kappa}{(1+\epsilon)n}\right)^{d\kappa}.$$

For $\kappa = 2$, the probability that there exists a pair of left vertices with a single neighbor is $\mathrm{O}(n^{3-2d})$. For any $3 \leq \kappa \leq \frac{n}{c}$, where c is an appropriately large constant (i.e. any $c > \exp\left(\frac{d}{d-2}\right)$ suffices), we can bound the probability P_κ by

$$\frac{e^{2\kappa}\kappa^{(d-2)\kappa}}{n^{(d-2)\kappa}}.$$

For any $3 \leq \kappa \leq \frac{n}{c}$, where c is chosen appropriately large, this quantity is non-increasing in κ and can be bounded by $\mathrm{O}(n^{3-2d})$, for $d \geq 3$.

For $\kappa > \frac{n}{c}$, let $\mu = \frac{\kappa}{n} \in (0,1]$. Using Proposition 1, we can bound the probability $P_{\mu n}$ by

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

To conclude the proof, it suffices to choose a value for d that makes the base of the above quantity strictly less than 1. By solving the resulting inequality, we obtain the following lower bound for d.

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

In order to simplify the above inequality, we observe that:

1. $\dfrac{\mu \ln\left(\frac{1}{\mu}\right)}{\mu \ln\left(\frac{1+\epsilon}{\mu}\right)} < 1$, $\forall \mu \in (0,1]$ and $\epsilon > 0$.
2. For small values of ϵ, $\dfrac{(1-\mu)\ln\left(\frac{1}{1-\mu}\right)}{\mu \ln\left(\frac{1+\epsilon}{\mu}\right)}$ is maximized for $\mu \approx \frac{1}{1+\epsilon}$, and for any $\epsilon \in (0,1)$, it cannot exceed $\dfrac{\epsilon \ln\left(\frac{1+\epsilon}{\epsilon}\right)}{\ln(1+\epsilon)}$.
3. $\dfrac{(1+\epsilon-\mu)\ln\left(\frac{1+\epsilon}{1+\epsilon-\mu}\right)}{\mu \ln\left(\frac{1+\epsilon}{\mu}\right)}$ is maximized for $\mu = 1$ achieving the value of $\dfrac{\epsilon \ln\left(\frac{1+\epsilon}{\epsilon}\right)}{\ln(1+\epsilon)}$.

Consequently, for any integer $d \geq 2 + \dfrac{2\epsilon \ln\left(\frac{1+\epsilon}{\epsilon}\right)}{\ln(1+\epsilon)}$, B contains a perfect matching with probability at least $1 - \mathrm{O}(n^{4-2d})$. A brief calculation using $\ln(1+\epsilon) > \epsilon - \epsilon^2/2$ shows that the above inequality is satisfied whenever $d \geq 2(1+\epsilon)\ln(\frac{e}{\epsilon})$. □

A.3 Proof of Lemma 2

We can think of the right vertices as bins and the random edges as balls. Each ball is placed into a bin selected uniformly at random and independently. Therefore, we have exactly $(1+\epsilon)\,n$ bins and dn balls.

The expected number of empty bins (i.e. isolated right vertices) is approximately $(1+\epsilon)ne^{-\frac{d}{1+\epsilon}}$ (e.g. [16]). If d is smaller than $(1+\epsilon)\ln\left(\frac{1+\epsilon}{(1+\beta)\epsilon}\right)$, the expected number of isolated right vertices is $(1+\beta)\epsilon n$. Furthermore, since the 0-1 random variables X_j corresponding to the events "right vertex j is isolated" are negatively associated (e.g. [9]), we can apply the Chernoff bound (e.g. [16], Chapter 4) to show that if $d < (1+\epsilon)\ln\left(\frac{1+\epsilon}{(1+\beta)\epsilon}\right)$, for some constant $\beta > 0$, then more than ϵn right vertices are isolated whp. Therefore, for such values of d, B does not contain a perfect matching whp. The lemma is obtained by setting $\beta = \epsilon$. □

A.4 Proof of Lemma 3

To prove Lemma 3, we need the following lemma whose proof is similar to Lemma 1.

Lemma 8. *Given a constant $\epsilon \in (0, 1/4)$, let δ be any positive constant not exceeding $\frac{4(1-4\epsilon)}{1+4\epsilon}$, and let d be any integer such that*

$$d \geq 3 + 2\delta + 2\epsilon(1+\delta) + \frac{(2+\delta)\epsilon \ln\left(\frac{1+\epsilon}{\epsilon}\right)}{\ln(1+\epsilon)}.$$

Then, any subset of left vertices X, $\frac{n}{2} \leq |X| \leq (1-(1+\delta)\epsilon)n$, has at least $(1+\epsilon)n - \frac{n-|X|}{1+\delta}$ neighbors whp.

Proof. Let μ be any number in the interval $\left[\epsilon, \frac{1}{2(1+\delta)}\right]$ such that the quantity $(1-(1+\delta)\mu)n$ is an integer. For simplicity of presentation and without loss of generality, we assume that $(1+\epsilon-\mu)n$ is also an integer.

In addition, let P_μ be the probability that there exists a set $X \subseteq L$ of cardinality $|X| = (1-(1+\delta)\mu)n$ which has less than $(1+\epsilon-\mu)n$ neighbors. We bound the probability that the graph B does not satisfy the conclusion of the lemma by the sum of the probabilities P_μ for all the different values of μ for which $(1-(1+\delta)\mu)n$ is an integer.

Having fixed an appropriate μ, let $X \subseteq L$, $|X| = (1-(1+\delta)\mu)n$, and let $Y \subseteq R$, $|Y| = (1+\epsilon-\mu)n$. The probability that the neighborhood of X is a subset of Y is

$$\mathbb{P}\mathrm{r}[\Gamma(X) \subseteq Y] = \left(\frac{1+\epsilon-\mu}{1+\epsilon}\right)^{d(1-(1+\delta)\mu)n}.$$

Since X can be chosen in $\binom{n}{(1-(1+\delta)\mu)n} = \binom{n}{(1+\delta)\mu n}$ different ways and Y can be chosen in $\binom{(1+\epsilon)n}{(1+\epsilon-\mu)n} = \binom{(1+\epsilon)n}{\mu n}$ different ways, the probability P_μ can be bounded from above by:

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

Using Proposition 1, we can bound the above product by

$$\left[\left(\frac{1}{1-(1+\delta)\mu}\right)^{1-(1+\delta)\mu}\left(\frac{1}{(1+\delta)\mu}\right)^{(1+\delta)\mu}\left(\frac{1+\epsilon}{1+\epsilon-\mu}\right)^{1+\epsilon-\mu}\left(\frac{1+\epsilon}{\mu}\right)^{\mu}\left(\frac{1+\epsilon-\mu}{1+\epsilon}\right)^{d(1-\mu(1+\delta))}\right]^n$$

Given ϵ and δ, the value of d should be chosen so as to ensure that the quantity in square brackets is strictly smaller than 1. Solving the resulting inequality yields the following lower bound for d:

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

For $\delta \in \left(0, \frac{4(1-4\epsilon)}{1+4\epsilon}\right]$, the right-hand side of Inequality (2) is maximized for $\mu = \epsilon$ yielding the following lower bound for d:

$$d > \frac{(1-\epsilon(1+\delta))\ln\left(\frac{1}{1-(1+\delta)\epsilon}\right) + (1+\delta)\epsilon\ln\left(\frac{1}{(1+\delta)\epsilon}\right) + \ln(1+\epsilon) + \epsilon\ln\left(\frac{1+\epsilon}{\epsilon}\right)}{(1-\epsilon(1+\delta))\ln(1+\epsilon)} \tag{3}$$

The right-hand side of Inequality (3) can be simplified to

$$d \geq \frac{1}{1-\epsilon(1+\delta)} + \frac{\ln\left(\frac{1}{1-(1+\delta)\epsilon}\right)}{\ln(1+\epsilon)} + \frac{(2+\delta)\epsilon\ln\left(\frac{1+\epsilon}{\epsilon}\right)}{\ln(1+\epsilon)} \tag{4}$$

by observing that $\ln\left(\frac{1}{(1+\delta)\epsilon}\right) < \ln\left(\frac{1+\epsilon}{\epsilon}\right)$.

To further simplify the right-hand side of Inequality (4), we observe that for $\delta \in \left(0, \frac{4(1-4\epsilon)}{1+4\epsilon}\right]$, the fraction $\frac{1}{1-(1+\delta)\mu}$ cannot exceed $1+2\epsilon(1+\delta)$ and that the fraction $\frac{\ln\left(\frac{1}{1-(1+\delta)\epsilon}\right)}{\ln(1+\epsilon)}$ cannot exceed $2(1+\delta)$.

Since $\mu \geq \epsilon$ and there are at most n different values of μ to consider, the probability that a graph B has the desired property is exponentially large, i.e. it is at least $1 - \mathrm{O}(n\beta^n)$, for some constant $\beta < 1$ depending on the choice of d. □

To conclude the proof of Lemma 3, we show that any graph B satisfying the conclusion of Lemma 8 has the desired property.

Let B be a graph satisfying the conclusion of Lemma 8. Suppose that for some μ, $\epsilon \leq \mu \leq \frac{1}{2(1+\delta)}$, there was a set $Y \subseteq R$ of size $|Y| = \mu n$ such that $\Gamma(Y) = X$ and $|X| < (1+\delta)\mu n$. Then the left vertices which are not neighbors of Y would comprise a set X' of cardinality strictly greater than $(1-(1+\delta)\mu)n$. By Lemma 8, the neighborhood of X' would be strictly greater than $(1+\epsilon-\mu)n$, which implies that some vertices of Y would be neighbors of X', a contradiction. □

A.5 Proof of Lemma 4

We prove the lemma by induction on λ. Recall than Y_λ denotes the set of right vertices at augmentation distance at most λ and X_λ denotes the subset of left vertices at most λ. In addition, $Y_0 = F$ and for any integer $\lambda \geq 0$, $X_{\lambda+1} = \Gamma(Y_\lambda)$ and $Y_{\lambda+1} = M(X_{\lambda+1}) \cup F$, where $M(X_{\lambda+1})$ denotes the mates of the vertices in $X_{\lambda+1}$.

Since we have assumed that any set $Y \subseteq R$, $\epsilon n \leq |Y| \leq \frac{n}{2(1+\delta)}$, expands by a factor of $(1+\delta)$ and $|Y_0| \geq \epsilon n$, at least $\epsilon(2+\delta)n$ right vertices are included in Y_1.

If for some λ, $1 \leq \lambda \leq \log_{(1+\delta)}\left(\frac{1}{2\epsilon}\right) - 1$, $|Y_\lambda| \geq \epsilon(1+(1+\delta)^\lambda)n$, then the cardinality of $Y_{\lambda+1}$ can be bounded from below by:

$$\begin{aligned}
|Y_{\lambda+1}| &= |X_{\lambda+1}| + |F| \\
&\geq (1+\delta)|Y_\lambda| + \epsilon n \\
&\geq \left[(1+\delta)(1+(1+\delta)^\lambda) + 1\right]\epsilon n \\
&\geq \epsilon(1+(1+\delta)^{\lambda+1})n
\end{aligned}$$

Since the expansion argument works until the size of Y_λ becomes larger than $(1/2+\epsilon)n$ for the first time, λ cannot exceed $\log_{(1+\delta)}\left(\frac{1}{2\epsilon}\right)$. □

Remark 1. We can actually prove that the number of right vertices at augmentation distance at most λ is at least $\epsilon((1+\delta)^{\lambda+1}-1)n$.

A.6 Proof of Lemma 5

To prove Lemma 5, we need the following lemma.

Lemma 9. *Let γ be any positive constant, and let $d \geq (1+\log e)(2+\gamma)+\log(1+\gamma)$ be an integer. Then, any subset of left vertices X, $|X| \leq \frac{n}{2(1+\gamma)}$, has at least $(1+\gamma)|X|$ neighbors with probability at least $1-\mathrm{O}(n^{3+\gamma-d})$.*

Proof. The proof is quite similar to the proof of Lemma 1. For some integer κ, $1 \leq \kappa \leq \frac{n}{2(1+\gamma)}$, we bound the probability P_κ that there exists some set $X \subseteq L$ of size κ which does not expand by $(1+\gamma)$. The probability that there exists a set X, $1 \leq |X| \leq \frac{n}{2(1+\gamma)}$, which does not expand by $(1+\gamma)$, is bounded by the sum of the probabilities P_κ for all different values of κ.

The probability P_κ that there exists a set X of size κ which has less than $(1+\gamma)\kappa$ neighbors can be bounded from above by

$$\binom{n}{\kappa}\binom{(1+\epsilon)n}{(1+\gamma)\kappa}\left(\frac{(1+\gamma)\kappa}{(1+\epsilon)n}\right)^{d\kappa} .$$

For small values of κ, i.e. $\kappa \in \{1,2,3\}$, we can easily bound P_κ by $\mathrm{O}(n^{2+\gamma-d})$. For $\kappa \in \left\{4,\ldots,\frac{n}{c}\right\}$, where $c>0$ is an appropriately large constant, the value of P_κ can be bounded from above by:

$$\frac{e^{\kappa(2+\gamma)}(1+\gamma)^{\kappa(d-(1+\gamma))}\kappa^{\kappa(d-(2+\gamma))}}{n^{\kappa(d-(2+\gamma))}} .$$

If c is chosen appropriately large, the above quantity cannot exceed $\mathrm{O}(n^{2+\gamma-d})$.

For $\kappa > \frac{n}{c}$, let $\mu = \frac{\kappa}{n} \in \left(0, \frac{1}{2(1+\gamma)}\right]$. Using Proposition 1, we can bound the probability P_κ by

$$\left[\left(\frac{1}{1-\mu}\right)^{1-\mu}\left(\frac{1}{\mu}\right)^{\mu}\left(\frac{1+\epsilon}{1+\epsilon-(1+\gamma)\mu}\right)^{1+\epsilon-(1+\gamma)\mu}\left(\frac{1+\epsilon}{(1+\gamma)\mu}\right)^{(1+\gamma)\mu}\left(\frac{(1+\gamma)\mu}{1+\epsilon}\right)^{d\mu}\right]^{n} .$$

It now suffices to choose a value for d that makes the base of the above quantity strictly less than 1. By solving the resulting inequality, we obtain the following lower bound for d.

$$d > 1+\gamma+\frac{\mu\ln\left(\frac{1}{\mu}\right)+(1-\mu)\ln\left(\frac{1}{1-\mu}\right)+(1+\epsilon-(1+\gamma)\mu)\ln\left(\frac{1+\epsilon}{1+\epsilon-(1+\gamma)\mu}\right)}{\mu\ln\left(\frac{1+\epsilon}{(1+\gamma)\mu}\right)} . \tag{5}$$

We observe that the right-hand side of Inequality (5) is maximized for $\mu = \frac{1}{2(1+\gamma)}$ and cannot exceed $(1+\log e)(2+\gamma)+\log(1+\gamma)$.

$\square$

To conclude the proof of Lemma 5, we show that any graph B satisfying the conclusion of Lemma 9 has the desired property.

Let B be a graph satisfying the conclusion of Lemma 9. Suppose that for some integer κ, $0 \leq \kappa \leq \frac{n}{2(1+\gamma)}$, there was a set $Y \subseteq R$, $|Y| = (1+\epsilon)n - (1+\gamma)\kappa$ such that $\Gamma(Y) = X$ and $|X| < n - \kappa$. Then the left vertices which are not neighbors of Y would comprise a set X' of cardinality strictly greater than κ. By Lemma 9, the neighborhood of X' would be strictly greater than $(1+\gamma)\kappa$, which implies that some vertices of Y would be neighbors of X', a contradiction. $\square$

A.7 Proof of Lemma 6

The proof is by induction on λ. By Lemma 4, we know that $|Y_{\lambda^\star}| \geq \frac{n(1/2+\epsilon)}{1+\epsilon}$. Therefore, for $\lambda = 0$, the number of right vertices at augmentation distance greater than $\lambda^\star$ is at most $\frac{n}{2}$.

For some integer $\lambda \geq 0$, let

$$|Y_{\lambda+\lambda^\star}| \geq (1+\epsilon)n - \frac{n}{2(1+\gamma)^\lambda} = (1+\epsilon)n - \frac{1}{2(1+\gamma)^{\lambda+1}}(1+\gamma)n \ .$$

By Lemma 5, the neighborhood of $Y_{\lambda+\lambda^\star}$ includes at least $n - \frac{1}{2(1+\gamma)^{\lambda+1}}$ left vertices, and hence the cardinality of $Y_{(\lambda+1)+\lambda^\star}$ is at least $(1+\epsilon)n - \frac{n}{2(1+\gamma)^{\lambda+1}}$. Consequently, at most $\frac{n}{2(1+\gamma)^{\lambda+1}}$ right vertices are at augmentation distance greater than $(\lambda+1)+\lambda^\star$. $\square$

A.8 Proof of Lemma 7

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 $(1 - \frac{1}{\ln(4/\gamma)})^{i-1} n \leq n_i \leq (1 - \frac{1}{\ln(4/\gamma)})^{i-1} + i$. 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 (1). 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 $\mathsf{free}(R_{i+1}) \leq \frac{\gamma}{2}|R_{i+1}|$ with probability $1 - O(\frac{1}{n})$. 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]:

$$\mathsf{free}(R_{i+1}) \leq \sum_{k=0}^{t/2} (-1)^k C_k \qquad (6)$$

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

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

Thus we get the following upper bound:

$$\mathbb{E}[\mathsf{free}(R_{i+1})] \leq (1 + O(\tfrac{1}{n_{i+1}}))\, n_{i+1} \sum_{k=0}^{t/2} (-1)^k \alpha^{k-1}/k! \leq \tfrac{3}{2} n_{i+1} e^{-\alpha}/\alpha \leq \tfrac{3\gamma}{8} |R_{i+1}| \tag{7}$$

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

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

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