Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 380 (2007) 47–68
Theoretical Computer Science
www.elsevier.com/locate/tcs

Balanced allocation and dictionaries with tightly packed constant size bins[☆]

Martin Dietzfelbinger[a,*], Christoph Weidling[b,1]

[a] *Technische Universität Ilmenau, 98684 Ilmenau, Germany*
[b] *Altova GmbH, Rudolfsplatz 13a, 1010 Wien, Austria*

Abstract

We study a particular aspect of the balanced allocation paradigm (also known as the "two-choices paradigm"): constant sized bins, packed as tightly as possible. Let $d \geq 1$ be fixed, and assume there are m bins of capacity d each. To each of $n \leq dm$ balls two possible bins are assigned at random. How close can $dm/n = 1 + \varepsilon$ be to 1 so that with high probability each ball can be put into one of the two bins assigned to it without any bin overflowing? We show that $\varepsilon > (2/e)^{d-1}$ is sufficient. If a new ball arrives with two new randomly assigned bins, we wish to rearrange some of the balls already present in order to accommodate the new ball. We show that on average it takes constant time to rearrange the balls to achieve this, for $\varepsilon > \beta^d$, for some constant $\beta < 1$. An alternative way to describe the problem is in data structure language. Generalizing cuckoo hashing [R. Pagh, F.F. Rodler, Cuckoo hashing, J. Algorithms 51 (2004) 122–144], we consider a hash table with m positions, each representing a bucket of capacity $d \geq 1$. Keys are assigned to buckets by two fully random hash functions. How many keys can be placed in these bins, if key x may go to bin $h_1(x)$ or to bin $h_2(x)$? We obtain an implementation of a dictionary that accommodates n keys in $m = (1+\varepsilon)n/d$ buckets of size $d = O(\log(1/\varepsilon))$, so that key x resides in bucket $h_1(x)$ or $h_2(x)$. For a lookup operation, only two hash functions have to be evaluated and two segments of d contiguous memory cells have to be inspected. If $d \geq 1 + 3.26 \cdot \ln(1/\varepsilon)$, a static arrangement exists with high probability. If $d \geq 16 \cdot \ln(1/\varepsilon)$, a dynamic version of the dictionary exists so that the expected time for inserting a new key is $\log(1/\varepsilon)^{O(\log\log(1/\varepsilon))}$.

Keywords: Randomized algorithms; Data structures; Dictionaries; Hashing; Random graphs; Balanced allocation; Storage space

1. Introduction: Bounded balanced allocation, d-orientability, and blocked cuckoo hashing

In this paper, we study a data allocation problem that can be described in different terminologies. Aspects of the problem have been considered in different contexts.

[☆] A preliminary version of this paper appeared in the Proceedings of ICALP 2005, Springer, LNCS, Vol. 3580, pp. 166–178.

[*] Corresponding address: Technische Universität Ilmenau, Institut fur Theoretische Informatik, Fachgebiet Komplexitatstheorie und Effiziente Algorithmen, P.O. Box 100565, 98684 Ilmenau, Germany. Tel.: +49 3677 69 2656; fax: +49 3677 69 1237.

E-mail addresses: martin.dietzfelbinger@tu-ilmenau.de (M. Dietzfelbinger), christoph.weidling@altova.com (C. Weidling).

[1] Affiliated with the Technische Universität Ilmenau while this work was done.

0304-3975/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2007.02.054

1.1. Bounded balanced allocation

In the **"balanced allocation paradigm"** (also known as the "two-choices paradigm") (Azar et al. [1]) we have n balls and m bins. To each ball, two bins are assigned at random. Each ball is to be placed into one of the two bins assigned to it; the aim is to keep the maximum load in the bins small. Much work has been devoted to analyzing the online version of this experiment, where the balls arrive one after the other and are put into a bin upon arrival, and this placement can never be changed later. Not very much is known about the offline version, where all balls with their two choices are given as input. In particular, it has been known for quite a while that for $n = O(m)$ with high probability[2] constant bin size is enough to place all balls. Some authors have considered "dynamic" situations in which the positions of balls may change in the course of a random process.

In this paper, we ask how well we can utilize the space in the bins if we fix the capacity of the bins to some number $d \geq 1$ (and assume that n and m are sufficiently large). We wish to keep the space overhead $\varepsilon = \frac{dm-n}{n}$ as small as possible and still be able to place the balls w.h.p. Our first main result is that $\varepsilon > (2/e)^{d-1}$ is sufficient to guarantee this. We also consider a dynamic version of the problem. Assume that n balls are placed and a new ball arrives, with two randomly assigned bins. We show that the expected time needed to rearrange the balls so that the new ball can be accommodated as well is constant, as long as $\varepsilon > \beta^d$, for some constant $\beta < 1$. This implies that the expected time to place n balls is $O(n)$. No estimates for the density that can be achieved asymptotically for fixed bin size d larger than some small constant have been known before, although the question arises in several applications.

1.2. d-Orientability for random graphs

It is easy to see that the allocation problem with bounded bin size d is equivalent to a version of the **"d-orientability problem"** for random graphs as formulated by Karp [14] (also see [2,24]). We say an undirected graph $G = (V, E)$ of m nodes and n edges is *d-orientable* if the edges can be directed in such a way that every node has outdegree not larger than d. Given a constant d, we ask for upper and lower bounds on the edge density $\frac{n}{m}$ (certainly $< d$) so that a graph with n randomly placed edges (including loops and multiple edges) is d-orientable w.h.p. Also, we ask how long it takes to adapt a given edge orientation when a new random edge arrives.

1.3. Blocked cuckoo hashing

Yet another formulation of the same problem can be given in **data structure** language. We wish to implement dynamic dictionaries so that constant lookup time is guaranteed. Dynamic dictionaries store keys from a universe U (possibly together with satellite data) and support the operations *insert*, *delete*, and *lookup*. Pagh and Rodler's **"cuckoo hashing"** method [21] assumes that each one of n keys is assigned to two locations $h_1(x)$ and $h_2(x)$ in a hash table of size m, and can be stored in one of the two locations. Each location has capacity 1. We generalize this approach by considering buckets of capacity d, for some arbitrary constant $d \geq 1$. Our construction results in the following. Assuming that fully random hash functions are available, we obtain an implementation of a dynamic dictionary that, for given $\varepsilon > 0$, stores n keys in space $(1 + \varepsilon)n$ in such a way that a lookup for x requires evaluating two hash functions and probing two blocks of d contiguous memory cells. The expected cost of inserting a new key is $(1/\varepsilon)^{O(\log\log(1/\varepsilon))}$. This compares favorably with the performance of "*d-ary cuckoo hashing*", a different generalization of cuckoo hashing by Fotakis, Pagh, Sanders, and Spirakis [10]. There $d = O(\log(1/\varepsilon))$ independent hash functions are used to achieve a similar space utilization. The access procedure of our scheme is more local and hence more suited for cache architectures. Experiments support the hope that the new scheme is competitive with d-ary cuckoo hashing [21] as far as space utilization is concerned, and allows faster accesses. (The interested reader may find a description of some of the experimental results in the Appendix.)

Remark 1. For reference, we describe the connection between the hashing formulation and the d-orientability formulation from [2,14,24] in detail. Given is a set S of n keys and two random hash functions h_1, h_2 with values in $[m] = \{0, \ldots, m-1\}$, the set of blocks. We consider a random (multi)graph $G_u = (V, E_u)$ with labeled edges. The

[2] In what follows, we write "w.h.p." for "with high probability", which means "with probability $1 - \frac{1}{\mathrm{poly}(n)}$".

node set is $V = [m]$, the set of labeled edges is $E_u = E_u(S, h_1, h_2) = \{\{h_1(x), h_2(x)\} \mid x \in S\}$. We say that G_u is *d-orientable* if the n edges in E_u can be directed in such a way that each node has outdegree at most d. Assigning such directions to edges is equivalent to storing the keys in a table with m blocks with maximal load d, as follows: the edge $\{y, y'\} = \{h_1(x), h_2(x)\}$ is directed from y to y' if and only if x is stored in block y. Below, any directed version of G_u with outdegree bounded by d will be called $G = (V, E)$.

Terminology in this paper. We have chosen to describe the algorithms and the analysis in the language of implementing a dictionary as a hash table with two functions. This also makes it possible to discuss a new solution to the problem involved in the assumption that fully random hash functions are available for free, see Section 2.3. All technical results readily translate into the terminology of the balanced allocation paradigm or the graph orientation paradigm.

1.4. Background and related work

Early contributions to our version of the allocation problem (fixed sized bins) were made in connection with the balanced allocation paradigm. (For a survey of this area, see [16].) In the seminal paper [1] by Azar, Broder, Karlin, and Upfal, it was noted ([11], also see [2]) that if a set of $n \le 1.67m$ balls is allocated to m cells, with two choices per ball, then with high probability the keys can be placed so that no bin holds more than two balls. This immediately extends to a scheme for storing n balls in m bins with a maximum load of $d = 2 \cdot \lceil n/(1.67m) \rceil \le 2 + 1.2n/m$, which for m, n large corresponds to a space overhead of $\varepsilon = 0.2$ in our notation.

Simultaneously, observations concerning the existence of such placements were made in papers on the simulation of parallel random access machines on distributed memory machines by redundantly storing data (e.g., [4,15]). In particular, it was shown there that a maximum load of $O(1 + n/m)$ is achievable with high probability, even if hash functions from classes described in [27] are used for allocating the bins.

Sanders [23,25] and Sanders et al. [26] studied the static allocation problem with fixed bin sizes as the combinatorial abstraction of a scheme called “Randomized Duplicate Allocation (RDA)”, used for storing data blocks on disks. In [26] it was shown that with high probability, a bin size of $d = 1 + \lceil n/m \rceil$ is sufficient (this would correspond to a bound $d \ge 1/\varepsilon$ in our notation). In [25], the question was asked how close n/m might get to $d = \lceil n/m \rceil$ so that still block size d is sufficient. For this, the criterion given as (20) in Section 3 was derived, and limiting values were determined by inspecting the corresponding functions for $d = 2, 3, \ldots, 9$ separately. No asymptotic relation between ε and d was derived. We close this gap, and moreover show how to calculate an assignment in expected linear time where [25,26] suggested using more expensive maxflow computations. In [23] an interesting algorithm (the “selfless algorithm”) was suggested that possibly (no proof provided) calculates an assignment for almost any random graph with not too large an edge density.

In [2] the online version of the case of heavily loaded bins (i.e., $n/m \to \infty$) was studied. In [5] it was demonstrated that perfect balance ($n = dm$, with no slack at all) is impossible w.h.p. if $d < \gamma_1 \ln n$ for a suitable constant γ_1, while perfect balance is possible w.h.p. if $d > \gamma_2 \ln n$ for some larger constant γ_2. For constant d and large n and m, no better bound than the one presented in [26] was given. Also in [5], a randomized rebalancing procedure was described and analyzed, with a running time polynomial in n.

On the data structures side, the paper by Pagh and Rodler [21] showed that $m = (2 + \varepsilon)n$ cells are enough if each cell may have load 1 and a key x may be stored in one of the locations given by two hash functions. It is not hard to see, using the threshold for the appearance of other than unicyclic components in random graphs, that it is impossible to decrease the space below $2n$ and still run cuckoo hashing in this original form.

As a remedy for this situation, Fotakis et al. [10] suggested “d-ary cuckoo hashing”. The scheme they study (in the language of implementing dictionaries) amounts to the balanced allocation problem with bin size 1 for the case where each ball has d random bins it can go to. Fotakis et al. [10] show that with n balls and $m = (1 + \varepsilon)n$ bins, it is necessary to have $d = \Omega(\log(1/\varepsilon))$ and sufficient to have $d = O(\log(1/\varepsilon))$ for it being possible to place the keys w.h.p., and that some $d = O(\log(1/\varepsilon))$ is sufficient to arrive at an insertion procedure that adds one new ball with d new random locations in expected constant time. This leads to an implementation of a dictionary for n keys in space $(1 + \varepsilon)n$, where a lookup requires evaluating d hash values and probing d random locations in the worst case, where $d = O(\log(1/\varepsilon))$. Inserting a key takes expected constant time. From the point of view of the efficient implementation of data structures, our result leads to a comparable space utilization, but has the advantage that only two hash functions have to be evaluated and two contiguous blocks of d memory cells must be probed in any search. (This approach is advantageous in architectures with caches, which could be observed also in experiments that we have conducted, see the Appendix. The different time requirements for searches would become clear if we counted the number of cache faults in the cache-oblivious model [12]. In comparison to d-ary cuckoo hashing, we get a reduction in the number of cache faults by a factor of $\Theta(B/\log|U|)$ where B is the cache block size and $\log|U|$ is the key length. We do not give the obvious details.)

Recently, Panigrahy [22] studied the dynamic version of the allocation problem (in the formulation for dynamic hash tables with insertions) for two choices and bin size $d = 2$. He established, by analyzing related branching processes, that inserting keys is possible in expected constant time as long as $n \leq 1.67m$ — the same bound as given for the static case in [1]. Our results are derived by totally different methods; they are not quite as tight for $d = 2$, but achieve a much better space utilization already if d is chosen only a little larger.

In the analysis of the static case, we start with the condition (20) on d and ε that has been noted already in [25]. The transformation of this condition into the general relation $d > 1 + \frac{\ln(1/\varepsilon)}{1-\ln 2}$ by means of calculus is a new contribution of this paper. Concerning the dynamic case, where we ask for the cost of adding one ball (or inserting one key), the basic structure of our analysis is the same as in [10], which in turn is based upon ideas from [17]. We show that the random graph generated by the n bins as nodes and the two possible locations for the n keys as undirected edges has certain expansion properties, and derive upper bounds on the probability that a random node is at a large distance to all non-full nodes in the directed version of the graph. In comparison to the analysis in [10], quite a few additional technical obstacles have to be overcome.

2. Detailed overview

In this section we describe the data structures, the algorithms, and the results of this paper.

2.1. The static case

A set S of n keys from the universe U is to be stored. We use an array (or "table") $T[0..m-1]$ consisting of $m = n(1+\varepsilon)/d$ blocks (subarrays) of d cells each. (For convenience, we assume that $n(1+\varepsilon)/d$ is an integer.) Inside each block, we store up to d keys sequentially. Given two hash functions $h_1, h_2 \colon U \to [m]$, we say that h_1, h_2 are *suitable* for S and d if it is possible to store each key x from S in one of the blocks $h_1(x), h_2(x)$ without any block receiving more than d keys. If the keys from S are stored according to h_1, h_2, a *lookup procedure* is obvious, which involves evaluating two hash values and searching two blocks. The first main result reads as follows.

Theorem 1. *Let $\varepsilon > 0$ be arbitrary. Assume that $d \geq 1 + \frac{\ln(1/\varepsilon)}{1-\ln 2}$. Let n be sufficiently large, let $S \subseteq U$ be an arbitrary set of n keys, and let T be a table with m blocks of size d each, where $dm \geq (1+\varepsilon)n$. Further assume that $h_1, h_2 \colon U \to [m]$ are fully random hash functions. Then, with probability $1 - O(1/m^{d-1})$ the functions h_1, h_2 are suitable for S and d.*

The bound on d stated in the theorem is not very far away from the true threshold. Indeed, the following can be shown by estimating the number of buckets hit by exactly $d-1$ hash values: If $d + 5\ln d + 6 < \frac{\ln(1/\varepsilon)}{1-\ln 2}$, then w.h.p. h_1, h_2 are *not* suitable for S. This result is given as Proposition 1 in Section 3.1.

The proof of Theorem 1 is presented in Section 3.2. It starts from a known characterization of h_1, h_2 being suitable [25,26], but in contrast with earlier approaches carries through a full analysis of the resulting estimates to obtain a closed bound valid for all d.

2.2. Updates: Insertion by BFS or the cuckoo procedure

Assume n keys are stored in the table T according to h_1, h_2, with blocks of size d.

Inserting a new key x can best be described in terms of the directed graph G from Remark 1. In G, find a directed path $y_0, y_1, \ldots, y_\ell$ with $y_0 \in \{h_1(x), h_2(x)\}$ and y_ℓ a node that is "*free*", i.e., has outdegree smaller than d. (This means that at least one of the d cells in block y_ℓ is empty.) The edges that form the path correspond to keys $x_1, \ldots, x_\ell$ such that x_i is stored in y_{i-1}, but may be stored in y_i. After moving x_i from y_{i-1} to y_i, for $1 \leq i \leq \ell$ (this corresponds to flipping the edges on the path), node (block) y_0 is free, and hence we can store x there.

We call a path $y_0, y_1, \dots, y_\ell$ as described an "augmenting path" for G and x. Obviously, if there is no augmenting path for G and x, the set Y_x of all nodes reachable in G starting from $\{h_1(x), h_2(x)\}$ consists only of full nodes. If we denote the set of $d \cdot |Y_x|$ keys stored in Y_x by S_x, then all $1 + d \cdot |Y_x|$ keys from $\{x\} \cup S_x$ need to be stored in Y_x, which is impossible. Turned the other way round, this means that if the keys from $S \cup \{x\}$ can be stored according to h_1, h_2 at all, then there is an augmenting path. So the problem is to find an augmenting path fast. As proposed in [10], a simple approach for this is *breadth-first-search* (*BFS*) in G, starting from $\{h_1(x), h_2(x)\}$. The time needed for this is proportional to the number of edges probed before a free node is found. Since the nodes in the part of G that is searched have outdegree d, this number is not larger than $2(d + d^2 + \cdots + d^\ell) < 4d^\ell$, where ℓ is the length of a *shortest* path from $\{h_1(x), h_2(x)\}$ to a free node. Thus we will have to analyze (the distribution of) the distance between $\{h_1(x), h_2(x)\}$ and the set of free nodes. Our second main result runs as follows.

Theorem 2. *Let $0 < \varepsilon \le 0.1$ be arbitrary. Assume that $d \ge 15.8 \cdot \ln(1/\varepsilon)$. Let n be sufficiently large, let S be an arbitrary set of n keys, let $x \in U - S$, and let T be a table with m blocks of size d each, where $dm \ge (1+\varepsilon)n$. Assume that $h_1, h_2 \colon U \to [m]$ are fully random hash functions, and that the keys from S have been stored in T by an algorithm that is ignorant of $h_1(x), h_2(x)$. Then the expected time needed to insert x by the BFS procedure is $(1/\varepsilon)^{O(\log d)}$.*

The proof is given in Section 4. (Some very technical parts are postponed to an extra section.)

Remark 2. The constant 15.8 in the bound is certainly not optimal; it appears as a consequence of some choices of the parameters and techniques used in the proofs. Informal observations on the basis of a central condition in the proof of Theorem 2, using a computer algebra system, indicate that for very small ε the relation $d \ge 4 \cdot \ln(1/\varepsilon)$ is sufficient. Such informal observations also lead to the conjecture that in the limit $\varepsilon \to 0$ the constant $1/(1 - \ln 2) \approx 3.26$ from the static case might be reached.

Remark 3. If the BFS is implemented naively, it could need $\Theta(n)$ extra space, which of course is undesirable. We do not address this issue in this paper, for the following reasons. First, it is possible, by means of some simple and not very interesting modifications to the straightforward BFS, to get by with $O(n^{1-\zeta})$ extra space in the worst case, for $\zeta > 0$ some constant. Second, if the technique described in Section 2.3 is used, no more than $n^{2/3}$ keys have to be handled at any time, so the scratch space problem vanishes altogether.

Remark 4 (*Insertion by Random Walk*). An alternative approach to insertion (also suggested in [10]) is to look for an augmenting path by a certain kind of random walk in G, as follows: Assume x is to be inserted. Repeat the following, starting with $z := x$:

> Calculate $h_1(z)$ and $h_2(z)$. If one of the two blocks $h_1(z)$ or $h_2(z)$ is not full, store z in one such block, and stop. (Ties are broken arbitrarily.) If both blocks are full, randomly choose one of the $2d$ keys stored in these blocks, call it z', kick z' out from its block and insert z in its place. (This is the "cuckoo step".)
> Let $z := z'$, and start again.

Many variations of this procedure are possible; e.g., one might try to insert z in the block it was not ejected from in the round before. Also, rules for stopping the loop have to be incorporated. All implementations used in the experiments we carried out (see the Appendix) are based on this random walk idea, not on the BFS procedure. It is an intriguing open problem to provide an analysis of the random walk insertion procedure, if possible establishing a bound of $O(1/\varepsilon)$ on the expected number of blocks probed in the course of an insertion. (The same question is open for d-ary cuckoo hashing.)

2.3. Sharing fully random hash functions

For the analysis to carry through, we assume that the hash functions h_1, h_2 behave fully randomly on S. If, in the course of inserting a key, it turns out that h_1, h_2 are not suitable, we might want to rehash the whole set, using new hash functions h_1, h_2. We cannot rely on the set S of keys to provide this degree of randomness. Note that although in the balanced allocation literature the full randomness assumption is routinely used, this is not the case in the hashing literature. Earlier work on hashing (e.g., [4,10,15,21]) has, very carefully, pointed out ways of working around this problem, for example by using functions from high-performance universal classes like in [27]. (This would not be sufficient for d-ary cuckoo-hashing, though.) In [9,19] it was demonstrated that full randomness can be simulated by universal hashing at the cost of $O(n)$ words of extra space. However, using such a construction would be unsatisfactory in our context, since we aim at getting by with εn extra space.

We propose the following workaround, which might be helpful also in other contexts. Let $\varepsilon > 0$ and n be given. Using high-performance hash classes [8,27] we may choose a function $h\colon U \to [n^{1/3}]$ so that with probability $1-n^{-c}$ (for some constant c), the set S is split into $n^{1/3}$ pieces $S_i = \{x \in S \mid h(x) = i\}$ of size $\leq (1+\frac{\varepsilon}{2})n^{2/3}$. It is not required that S is known or the pieces S_i are listed. For each of the pieces S_i, we run cuckoo hashing with blocks of size d in a separate table T_i of size $(1+\varepsilon)n^{2/3}$. It is an easy exercise, using some polynomial hash functions and techniques described in [8], to provide a pair h_1, h_2 of hash functions that with high probability behaves fully randomly on one S_i, if we are allowed to use space $n^{5/6}$ for storing h_1, h_2. Since we may use the *same* hash function pair h_1, h_2 for all pieces S_i, $i = 0, \ldots, n^{1/3} - 1$, the overall space needed for the fully random functions is $O(n^{5/6}) = o(n)$. The algorithms and arguments described in the present paper then have to be applied to each of the pieces separately. (Details and other applications of this "splitting trick" will be described in a forthcoming paper [7].)

2.4. *Some inequalities*

For later use, we state some (standard) inequalities. We will be using the following upper bound for binomial coefficients:

$$\binom{n}{k} \leq \frac{n^n}{k^k (n-k)^{n-k}} = \left(\frac{1}{\mu^\mu (1-\mu)^{1-\mu}}\right)^n, \quad \text{for } 0 \leq k \leq n, \tag{1}$$

where $\mu = k/n$.

Proof. By the binomial formula:

$$1 = (\mu + (1-\mu))^n \geq \binom{n}{k}\mu^k (1-\mu)^{n-k} = \binom{n}{k}(\mu^\mu (1-\mu)^{1-\mu})^n. \quad \square$$

If k is small in comparison to n, the following weaker inequality is already helpful:

$$\binom{n}{k} \leq \left(\frac{e \cdot n}{k}\right)^k. \tag{2}$$

(This is immediate from $\binom{n}{k} \leq n^k/k!$ and $k^k/k! \leq \sum_{i\geq 0} k^i/i! = e^k$.)

Further, a standard version of the Chernoff–Hoeffding bounds is used repeatedly: If $X_1, \ldots, X_n$ are independent 0-1-valued random variables and $X = X_1 + \cdots + X_n$, then for $\mathbf{E}(X) \leq a \leq n$ we have

$$\mathbf{Prob}(X \geq a) \leq \left(\frac{\mathbf{E}(X)}{a}\right)^a \left(\frac{n - \mathbf{E}(X)}{n-a}\right)^{n-a}. \tag{3}$$

(For a proof, see e.g. [13].)

3. Analysis for the static case

In this section, we analyze the size of d needed for storing a set S of n keys in $m = (1+\varepsilon)n/d$ blocks of size d, if $h_1, h_2\colon U \to [m]$ are fully random. We first show that asymptotically (for $\varepsilon \to 0$) the factor $\frac{1}{1-\ln 2}$ in Theorem 1 is the correct threshold. The (involved) proof of Theorem 1 follows afterwards.

3.1. *A lower bound on d*

Proposition 1. *For $\varepsilon > 0$ small enough, we have the following:*
If $6+5\ln d+d < \frac{\ln(1/\varepsilon)}{1-\ln 2}$ and $n, m \to \infty$, then with high probability random hash functions h_1 and h_2 are not suitable for a set S of n keys.

Proof. We fix one bucket $y \in [m]$ and estimate the probability p_{d-1} that exactly $d-1$ of the $2n$ random hash values $\{h_1(x), h_2(x) \mid x \in S\}$ equals y. The number of hits is binomially distributed, hence

$$p_{d-1} = \binom{2n}{d-1} \cdot \frac{1}{m^{d-1}} \cdot \left(1 - \frac{1}{m}\right)^{2n-(d-1)}. \tag{4}$$

We simplify, using that $dm = (1+\varepsilon)n$ and that $\binom{2n}{d-1} = \frac{(2n)^{d-1}}{(d-1)!}(1 - O(1/n))$ and that $(1 - 1/m)^{2n} = (1 - 1/m)^{m \cdot 2d/(1+\varepsilon)}$:

$$p_{d-1} \geq 2^{d-1} \cdot \frac{d^d}{d!} \cdot \frac{1}{(1+\varepsilon)^{d-1}} \cdot e^{-2d/(1+\varepsilon)} \cdot (1 - O(1/n)). \tag{5}$$

Using the fact that $d! < \sqrt{2\pi d}(d/e)^d \cdot e^{1/12d} < 2.6 \cdot (d/e)^d\sqrt{d}$ and that $1/(1+\varepsilon) \leq 1 - \varepsilon/2$ we get

$$p_{d-1} \geq 2^{d-1} \cdot \frac{e^d}{2.6\sqrt{d}} \cdot \frac{1}{(1+\varepsilon)^{d-1}} \cdot e^{-2d+\varepsilon d} \cdot (1 - O(1/n)) = \frac{(2/e)^d}{5.2\sqrt{d}} \cdot (1 - O(1/n)). \tag{6}$$

By the linearity of expectations, the expected number of buckets that are hit by exactly $d-1$ hash values is

$$p_{d-1} \cdot m \geq m \cdot ((2/e)^{-d}/5.2\sqrt{d}) \cdot (1 - O(1/n)). \tag{7}$$

If this expected value is larger by a constant factor than εn (e.g., $mp_{d-1} \geq 1.1\varepsilon n$), then with high probability more than εn buckets are hit by exactly $d-1$ hash values. (This follows by applying standard tail inequalities like the Azuma–Hoeffding inequality [18, pp. 91–96].) These buckets cannot be full in any arrangement, so there are more than εn non-full buckets. This is impossible, so such h_1 and h_2 are not suitable. By (6), a sufficient condition for this situation is:

$$\frac{(2/e)^d}{5.2\sqrt{d}} \cdot m > 1.1\varepsilon n. \tag{8}$$

Substituting $dm = (1+\varepsilon)n$ and rearranging we see that (8) is equivalent to $\varepsilon/(1+\varepsilon) < (2/e)^d/(5.72 d^{3/2})$. Taking logarithms we obtain that the following is sufficient for h_1 and h_2 being not suitable w.h.p.:

$$(1 - \ln 2) \cdot d + \ln(5.72) + \frac{3 \ln d}{2} < \ln\left(\frac{1}{\varepsilon}\right). \tag{9}$$

Since $\ln(5.72)/(1 - \ln 2) < 6$ and $\frac{3}{2}/(1 - \ln 2) < 5$, the proposition follows. $\square$

3.2. *Proof of Theorem 1*

For proving Theorem 1, we set out as in [10,25,26], observing a basic necessary and sufficient condition for h_1, h_2 being suitable for S. For $X \subseteq S$ let $\Gamma(X) = \{h_1(x), h_2(x) \mid x \in X\}$.

Fact 1. *Let h_1 and h_2 be hash functions mapping S to $[m]$. Then S can be stored in m blocks of size d according to h_1 and h_2 if and only if for every $X \subseteq S$ we have $|\Gamma(X)| \geq \frac{1}{d}|X|$.*

(For the convenience of the reader, we include a *proof* of Fact 1: Form the bipartite graph $G' = (S, [m] \times [d], E')$, with

$$E' = \{(x, (h_i(x), j)) \mid x \in S, i \in \{1, 2\}, 0 \leq j < d\}.$$

The set $[m] \times [d]$ of nodes on the right side provides d copies for each block $y \in [m]$; each $x \in S$ is connected to the copies of the blocks $h_1(x)$ and $h_2(x)$. It is then obvious that h_1, h_2 are suitable for S if and only if G' has a matching that covers all nodes in S. By Hall's marriage theorem [6, p. 31], this is the case if and only if in G' each set $X \subseteq S$ has at least $|X|$ neighbors in $[m] \times [d]$. Clearly, this holds if and only if $|\Gamma(X)| \geq |X|/d$ for each set $X \subseteq S$.)

Fact 1 implies that to prove Theorem 1 it is sufficient to establish an upper bound on

$$F := \mathbf{Prob}(\exists X \subseteq S\colon |\Gamma(X)| < |X|/d).$$

Lemma 1. *If* $\varepsilon \le 0.25$ *and* $d > 1 + \frac{\ln(1/\varepsilon)}{1-\ln 2}$, *then* $F = O(m^{1-d})$.

Proof. For $1 \le j \le m/(1+\varepsilon)$, let $F(j)$ be the probability that there is a set Y of j blocks and a set $X \subseteq S$ of dj keys which satisfies $\Gamma(X) \subseteq Y$. Clearly,

$$F \le \sum_{1\le j\le m/(1+\varepsilon)} F(j). \tag{10}$$

To bound $F(j)$, we apply the Chernoff–Hoeffding bound (3). For a fixed $Y \subseteq [m]$ of size j, define random variables I_x, $x \in S$, as follows:

$$I_x := \begin{cases} 1, & \text{if } h_1(x), h_2(x) \in Y, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Further let $I = \sum_{x\in S} I_x$. We have $\mathbf{Prob}(I_x = 1) = (j/m)^2$ and $\mathbf{E}(I) = \sum_{x\in S}\mathbf{E}(I_x) = n\frac{j^2}{m^2}$. Because the I_x are independent, by (3) we obtain the following bound:

$$\mathbf{Prob}(I \ge jd) \le \left(\frac{nj^2}{m^2 jd}\right)^{jd} \left(\frac{n(m^2-j^2)}{m^2(n-jd)}\right)^{n-jd}. \tag{12}$$

Because there are $\binom{m}{j}$ sets Y of size j, we get

$$F(j) \le \binom{m}{j} \left(\frac{nj}{m^2 d}\right)^{jd} \left(\frac{n(m^2-j^2)}{m^2(n-jd)}\right)^{n-jd}. \tag{13}$$

We apply (1) and substitute $n = \frac{dm}{1+\varepsilon}$ to obtain

$$\begin{aligned} F(j) &\le \frac{m^m}{j^j(m-j)^{m-j}} \left(\frac{j}{(1+\varepsilon)m}\right)^{jd} \left(\frac{d(m^2-j^2)}{(1+\varepsilon)m(n-jd)}\right)^{dm/(1+\varepsilon)-jd} \\ &= \frac{m^m}{j^j(m-j)^{m-j}} \left(\frac{j}{(1+\varepsilon)m}\right)^{jd} \left(\frac{m^2-j^2}{(1+\varepsilon)m(m/(1+\varepsilon)-j)}\right)^{dm/(1+\varepsilon)-jd} \\ &= (1+\varepsilon)^{-jd} m^{\frac{m(1+\varepsilon-d)}{1+\varepsilon}} j^{j(d-1)} (m-j)^{j-m} \left(\frac{m^2-j^2}{m-j(1+\varepsilon)}\right)^{d\frac{m-j(1+\varepsilon)}{1+\varepsilon}}. \end{aligned} \tag{14}$$

Inequality (14) was already observed in [25,26], and used for finding sufficient conditions on ε for bucket sizes $d = 2, 3, \ldots, 9$. We proceed to establish asymptotic bounds. We abbreviate the expression of the right-hand side of (14) as $f(j,\varepsilon)$ and estimate $F(j)$ or $f(j,\varepsilon)$ in different ranges of j, $1 \le j \le m/(1+\varepsilon)$.

Case 1: $j = 1$. — By (13) we get:

$$\begin{aligned} F(1) &\le m\left(\frac{1}{(1+\varepsilon)m}\right)^d \left(\frac{n-n/m^2}{n-d}\right)^{n-d} < m\left(\frac{1}{(1+\varepsilon)m}\right)^d e^{d-n/m^2} \\ &= O\left(\frac{1}{m^{d-1}}\right). \end{aligned} \tag{15}$$

Case 2: $2 \le j < m/(2e^4)$. — We first note that for j fixed $f(j,\varepsilon)$ is decreasing in ε. For this, we differentiate (using a computer algebra system):

$$\frac{\partial \ln(f(j,\varepsilon))}{\partial\varepsilon} = -d(1+\varepsilon)^{-2} m \ln\left(\frac{1-(j/m)^2}{1-j(1+\varepsilon)/m}\right). \tag{16}$$

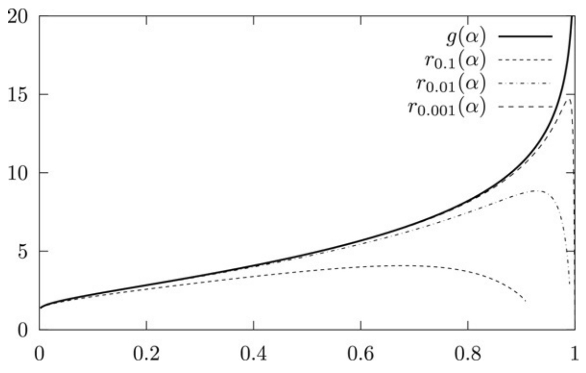


Fig. 1. The graphs of g and r_ε for several ε.

Since $j/m < 1$, we have $1-(j/m)^2 > 1-(1+\varepsilon)j/m$; hence $\frac{\partial}{\partial\varepsilon}\ln(f(j,\varepsilon)) < 0$. This means that $f(j,\varepsilon) < f(j,0)$, and we may continue from (14) as follows:

$$\begin{aligned} F(j) < f(j,0) &= m^{m(1-d)} j^{j(d-1)} (m-j)^{j-m} (m+j)^{d(m-j)} \\ &= m^{m(1-d)} j^{j(d-1)} m^{j-m} \left(1-\frac{j}{m}\right)^{j-m} m^{d(m-j)} \left(1+\frac{j}{m}\right)^{d(m-j)} \\ &< m^{j(1-d)} j^{j(d-1)} e^{\frac{j(m-j)}{m}} e^{\frac{dj(m-j)}{m}} < \left(\left(\frac{j}{m}\right)^{d-1} e^{d+1}\right)^{j}. \end{aligned} \tag{17}$$

The terms $((\frac{j}{m})^{d-1}e^{d+1})^j$ are geometrically decreasing for $2 \le j < m/(2e^4)$; hence (using (14)):

$$\sum_{2\le j<m/(2e^4)} F(j) = O(f(2,0)) = O\left(\left(\left(\frac{2}{m}\right)^{d-1} e^{d+1}\right)^2\right) = O\left(\frac{1}{m^{2d-2}}\right). \tag{18}$$

In preparation for the remaining two cases, we substitute $\alpha = j/m$ in the right-hand side $f(j,\varepsilon)$ of (14) and take logarithms. Using the fact that $\alpha \le \frac{1}{1+\varepsilon}$ we obtain that $\ln(F(j)) \le mR_\varepsilon(\alpha)$, where

$$\begin{aligned} R_\varepsilon(\alpha) &:= -\alpha\ln\alpha - (1-\alpha)\ln(1-\alpha) + \alpha d(\ln\alpha - \ln(1+\varepsilon)) \\ &\quad + \frac{d(1-\alpha(1+\varepsilon))(\ln(1-\alpha^2) - \ln(1-\alpha(1+\varepsilon)))}{1+\varepsilon}. \end{aligned} \tag{19}$$

If d is chosen so that $R_\varepsilon(\alpha) < 0$ for $e^{-4} \le \alpha \le \frac{1}{1+\varepsilon}$, we will have $F(j) \le \gamma^m$ for a constant $\gamma < 1$, for all j in the considered range, which will be sufficient to bound the sum of all these $F(j)$'s by $m\cdot\gamma^m$, which is vanishingly small. In order to find a suitable d, we rearrange the expression for $R_\varepsilon(\alpha)$ and get the following sufficient condition for $R_\varepsilon(\alpha) < 0$:

$$d > \frac{\alpha\ln\alpha + (1-\alpha)\ln(1-\alpha)}{\alpha(\ln\alpha - \ln(1+\varepsilon)) + \frac{(1-\alpha(1+\varepsilon))(\ln(1-\alpha^2)-\ln(1-\alpha(1+\varepsilon)))}{1+\varepsilon}} =: r_\varepsilon(\alpha). \tag{20}$$

We must find a simple upper bound on $r_\varepsilon(\alpha)$. It is obvious that for each fixed α and $\varepsilon \to 0$, the values $r_\varepsilon(\alpha)$ converge to

$$g(\alpha) := \frac{\alpha\ln\alpha + (1-\alpha)\ln(1-\alpha)}{\alpha\ln\alpha + (1-\alpha)\ln(1+\alpha)}. \tag{21}$$

As a first step, we show that this convergence is monotone from below, see Fig. 1.

Claim: The function $g(\alpha)$ is an upper bound for $r_\varepsilon(\alpha)$, for all $\varepsilon \le 0.5$ and all α, $0 \le \alpha \le \frac{1}{1+\varepsilon}$.

Proof of claim: The expressions $r_\varepsilon(\alpha)$ and $g(\alpha)$ are fractions of the form A/B and A/B', where $A, B, B' < 0$. This implies that $A/B \le A/B'$ if and only if $B \le B'$. That means that to show that $r_\varepsilon(\alpha) \le g(\alpha)$, we must prove the

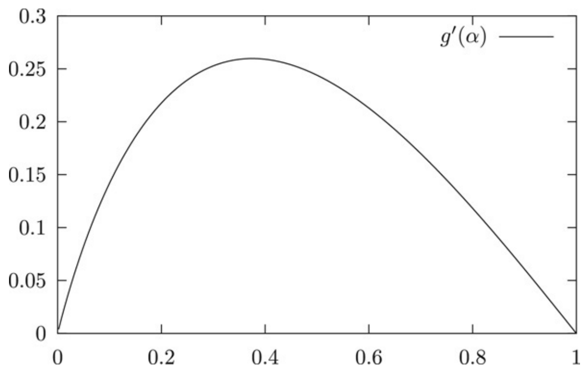


Fig. 2. The graph of the numerator of g'.

following inequality:

$$\begin{aligned} 0 &\le \alpha\ln\alpha + (1-\alpha)\ln(1+\alpha) - \left(\alpha(\ln\alpha - \ln(1+\varepsilon)) + \frac{(1-\alpha(1+\varepsilon))(\ln(1-\alpha^2) - \ln(1-\alpha(1+\varepsilon)))}{1+\varepsilon}\right) \\ &= (1-\alpha)\ln(1+\alpha) + \alpha\ln(1+\varepsilon) - \left(\frac{1}{1+\varepsilon} - \alpha\right)\ln\left(\frac{1-\alpha^2}{1-\alpha(1+\varepsilon)}\right). \qquad (22) \end{aligned}$$

This is easy if we fix $\alpha < 1$ and vary ε with $(1+\varepsilon)\alpha < 1$: The derivative of the expression in (22) with respect to ε is

$$\frac{\ln(1-\alpha^2) - \ln(1-\alpha(1+\varepsilon))}{(1+\varepsilon)^2} = \frac{\ln\left(\frac{1-\alpha^2}{1-\alpha(1+\varepsilon)}\right)}{(1+\varepsilon)^2} > 0. \qquad (23)$$

Therefore the expression in (22) is minimal for $\varepsilon = 0$, which means:

$$\begin{aligned} &(1-\alpha)\ln(1+\alpha) + \alpha\ln(1+\varepsilon) - \left(\frac{1}{1+\varepsilon} - \alpha\right)\ln\left(\frac{1-\alpha^2}{1-\alpha(1+\varepsilon)}\right) \\ &\quad \ge (1-\alpha)\ln(1+\alpha) - (1-\alpha)\left(\ln(1-\alpha^2) - \ln(1-\alpha)\right) = 0, \qquad (24) \end{aligned}$$

which shows that (22) is true, and proves the claim.

Now we proceed to the two remaining cases.

Case 3: $m/(2e^4) \le j \le (1-2\varepsilon)m$. — By the claim, we need to analyze the function $g(\alpha)$ in the interval $[1/(2e^4), 1-2\varepsilon]$. Now g is a fixed differentiable function (see Fig. 1), with the following derivative:

$$g'(\alpha) = \frac{\ln(\alpha)((1+\alpha)\ln(1+\alpha) - 2\alpha) + \ln(1-\alpha)(2\alpha - (1+\alpha)\ln(\alpha) - 2)}{(\alpha\ln(\alpha) + \ln(1+\alpha) - \ln(1+\alpha)\alpha)^2(1+\alpha)}. \qquad (25)$$

Clearly, the denominator of this expression is positive for $0 < \alpha < 1$, so the numerator determines the sign of $g'(\alpha)$. We spare the reader a tedious exercise in calculus by pointing to a plot of this numerator (see Fig. 2), which makes it clear that it is positive as well.

This means that $g(\alpha)$ is strictly increasing and, using the claim, for $\alpha \le 1-2\varepsilon$ we have

$$r_\varepsilon(\alpha) \le g(\alpha) \le g(1-2\varepsilon) = 1 + \frac{\ln\varepsilon - \ln(1-\varepsilon)}{\frac{(1-2\varepsilon)\ln(1-2\varepsilon)}{2\varepsilon} + \ln 2 + \ln(1-\varepsilon)}. \qquad (26)$$

Our next aim is to bound $g(1-2\varepsilon)$ from (26).

We want to show that $g(1-2\varepsilon) \le 1 + \frac{-\ln\varepsilon}{1-\ln 2}$. A plot of the function $1 + \frac{-\ln\varepsilon}{1-\ln 2} - g(1-2\varepsilon)$ (see Fig. 3) indicates that this difference is positive for $0 < \varepsilon < 0.5$.

A rigorous proof for $0 < \varepsilon \le 0.25$ could run along the following lines. The inequality $g(1-2\varepsilon) \le 1 + \frac{-\ln\varepsilon}{1-\ln 2}$ holds if and only if

$$\left(\frac{(1-2\varepsilon)\ln(1-2\varepsilon)}{2\varepsilon} + 1 + \ln(1-\varepsilon)\right)\ln\varepsilon - (1-\ln 2)\ln(1-\varepsilon) \ge 0. \qquad (27)$$

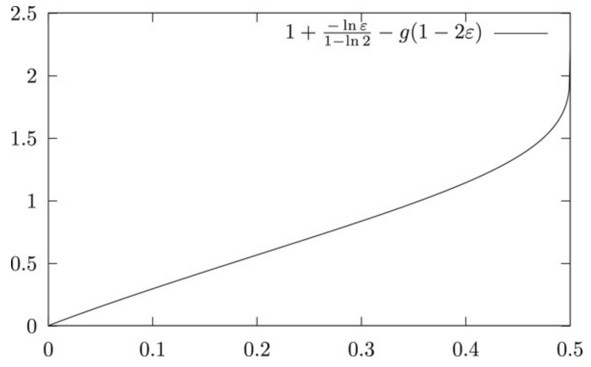


Fig. 3. The graph of $1+\frac{-\ln\varepsilon}{1-\ln 2}-g(1-2\varepsilon)$.

Consider the Taylor series of $\frac{(1-2\varepsilon)\ln(1-2\varepsilon)}{2\varepsilon}+1+\ln(1-\varepsilon)$ around $\varepsilon_0=0$, which turns out to be

$$\sum_{k\geq 2}\frac{1}{k}\Big(\frac{2^k}{k+1}-1\Big)\varepsilon^k. \tag{28}$$

For $\varepsilon\leq 0.25$, the terms in this series decrease geometrically by a rate of more than a factor of 2, so the sum is bounded by $2\cdot\frac{1}{2}(\frac{4}{3}-1)\varepsilon^2=\varepsilon^2/3$. Hence the left hand side in (27) is bounded below by $\frac{1}{3}\varepsilon^2\ln\varepsilon+(1-\ln 2)\varepsilon$. Since $\varepsilon\ln\varepsilon$ is (negative and) decreasing in $[0,0.25]$, we obtain $\frac{1}{3}\varepsilon^2\ln\varepsilon+(1-\ln 2)\varepsilon\geq(\frac{1}{3}\varepsilon\ln\varepsilon+(1-\ln 2))\varepsilon>0$ for ε in this range.

We summarize: If we choose

$$d\geq 1+\frac{1}{1-\ln 2}\ln\left(\frac{1}{\varepsilon}\right)=1+3.25889\ldots\ln\left(\frac{1}{\varepsilon}\right), \tag{29}$$

then $F(j)=O(\gamma^n)$ for a constant $\gamma<1$, for $m/(2e^4)\leq j\leq(1-2\varepsilon)m$.

Case 4: $(1-2\varepsilon)m\leq j\leq m/(1+\varepsilon)$. — Let $0\leq\varepsilon\leq 0.25$ be fixed. Our aim is to show that $r_\varepsilon(\alpha)$ as a function of α is decreasing in the interval $[1-2\varepsilon,\frac{1}{1+\varepsilon}]$, and hence $r_\varepsilon(1-2\varepsilon)$ can serve as an upper bound. For this, we wish to show that the derivative $r'_\varepsilon(\alpha)$ is negative in $[1-2\varepsilon,\frac{1}{1+\varepsilon}]$.

As a first step, consider r'_ε at $\alpha=1-2\varepsilon$:

$$r'_\varepsilon(1-2\varepsilon)=\frac{((1-2\varepsilon)\ln(1-2\varepsilon)+2\varepsilon\ln\varepsilon)(2\ln 2+\ln(1-\varepsilon)-\ln(1+2\varepsilon))}{\left((1-\varepsilon-\varepsilon^2)\ln\left(\frac{1+\varepsilon}{1-2\varepsilon}\right)-(2\varepsilon^2+\varepsilon)\ln\left(\frac{4(1-\varepsilon)}{1+2\varepsilon}\right)\right)^2}. \tag{30}$$

We show that the numerator of the fraction in (30) is negative, and hence that $r'_\varepsilon(1-2\varepsilon)<0$.

For this, we examine the two factors of the numerator. Obviously, $(1-2\varepsilon)\ln(1-2\varepsilon)+2\varepsilon\ln\varepsilon<0$. By $\ln(1-x)\geq -x/(1-x)$, $0\leq x<1$, we get for $\varepsilon\leq 0.25$:

$$2\ln 2+\ln(1-\varepsilon)-\ln(1+2\varepsilon)=2\ln 2+\ln\left(1-\frac{3\varepsilon}{1+2\varepsilon}\right)>2\ln 2-\frac{3\varepsilon}{1-\varepsilon}>0. \tag{31}$$

Next, we wish to show that r'_ε is decreasing in $[1-2\varepsilon,\frac{1}{1+\varepsilon}]$. For this, we must understand the behaviour of the second derivative of r_ε. Let us denote the denominator of $r_\varepsilon(\alpha)$ in (20) by u. We differentiate twice to obtain

$$u''=\frac{2\alpha(1+\alpha^2)(2\varepsilon+\varepsilon^2+2)-(1+\varepsilon)(6\alpha^2+1+\alpha^4)}{\alpha(\alpha^2-1)^2(\varepsilon\alpha+\alpha-1)(1+\varepsilon)}. \tag{32}$$

The numerator of u'' is a polynomial of degree four. We discuss the position of its roots. Since $\varepsilon<0.25$, there is only one root in $(0,1/(1+\varepsilon)]$, which is $1+\varepsilon-\sqrt{\varepsilon^2+2\varepsilon}$. Further we have

$$1+\varepsilon-\sqrt{\varepsilon^2+2\varepsilon}<1-2\varepsilon, \tag{33}$$

Therefore u'' does not have a root in $[1-2\varepsilon, 1/(1+\varepsilon)]$. At $\alpha = 1-2\varepsilon$, we find

$$u''(1-2\varepsilon) = -\frac{(1-4\varepsilon)(2\varepsilon^2+1)}{4\varepsilon(1-\varepsilon)^2(1+\varepsilon)(1-4\varepsilon^2)} < 0. \tag{34}$$

Hence u'' is negative if $1-2\varepsilon \le \alpha \le 1/(1+\varepsilon)$.

The second derivative $\frac{1}{\alpha(1-\alpha)}$ of the numerator v of r_ε is positive. Both u and v are negative. This implies

$$(v'u - vu')' = v''u - vu'' < 0, \tag{35}$$

and hence $v'u - vu'$ is decreasing. Together with the fact that $v'u - vu'$ is negative in $\alpha = 1-2\varepsilon$, this shows that $v'u - vu' < 0$ in $[1-2\varepsilon, 1/(1+\varepsilon)]$. Thus

$$r'_\varepsilon = \left(\frac{v}{u}\right)' = \frac{v'u - vu'}{u^2} < 0, \tag{36}$$

and hence r_ε is decreasing. Hence $r_\varepsilon(\alpha) \le r_\varepsilon(1-2\varepsilon)$ for $1-2\varepsilon \le \alpha \le \frac{1}{1+\varepsilon}$, and we get $F(j) = O(\gamma^n)$ as in Case 3. This finishes the proof of Lemma 1. □

It is clear that Lemma 1 implies Theorem 1 for the case $0 < \varepsilon \le 0.25$. How about larger values of ε than 0.25? We first note that as soon as $d > 1 + \ln(1/0.25)/(1-\ln 2) \approx 5.52$, we may apply Lemma 1 to $\varepsilon = 0.25$ and use the monotonicity property of $r_\varepsilon(\alpha)$ with respect to ε, as established in the proof of the "Claim". The only cases not covered are $d = 1, \dots, 5$.

For the special case $d = 1$, we refer to random graph theory [3] to note that any $\varepsilon > 1$ is sufficient. For the values $d = 2, 3, 4, 5$, we use the method from [25]: Fix d, and utilizing a computer algebra system show that some $\varepsilon(d) < 1 + \ln(1/\varepsilon)/(1-\ln 2)$ satisfies $r_{\varepsilon(d)}(\alpha) < d$ for $0 \le \alpha < 1/(1+\varepsilon(d))$. By the monotonicity property of $r_\varepsilon(\alpha)$, then $r_\varepsilon(\alpha) < d$ for all $\varepsilon \ge \varepsilon(d)$ as well. It turns out that, for example, $\varepsilon(2) = 0.6$, $\varepsilon(3) = 0.25$, $\varepsilon(4) = 0.11$, and $\varepsilon(5) = 0.065$ are suitable.

4. The expected insertion time

In this section, we analyze the expected time of the BFS algorithm for inserting a new key x in T, as described in Section 2.2 (also recall Remark 1). Just before x is inserted, a set S of n keys is stored in T. We assume that the directed graph G determined by this placement (see Remark 1) is independent of the hash values $h_1(x), h_2(x)$ — this is the case if x was never inserted before. We start a BFS in G from $\{h_1(x), h_2(x)\}$ with the aim of finding a shortest path to a node in

$$Y_0 := \{y \in V \mid y \text{ is free in } G\}.$$

The time for the BFS in G is $O(\min\{n, d^{\ell+1}\})$, where ℓ is the length of such a shortest path. Our aim is to verify that the expectation of the number of edges we have to inspect before a free node is found is $O(1)$. For this, we analyze the distribution of the number of nodes at different distances to Y_0. (The analysis runs along the lines of that of [10], but we are dealing with quite a different graph.) — Recursively define, for $k \ge 1$:

$$\begin{aligned} X_k &:= \{x \in S \mid h_1(x) \in Y_{k-1} \text{ or } h_2(x) \in Y_{k-1}\} \text{ and} \\ Y_k &:= Y_{k-1} \cup \{y \mid \exists x \in X_k, i \in \{1,2\}\colon x \text{ is stored in block } y = h_i(x)\}. \end{aligned}$$

We say that the keys of X_k "*hit*" Y_{k-1}. By the definition, $Y_{k-1} \subseteq Y_k$, for $k = 1, 2, \dots$. It is easy to see that Y_k is the set of nodes from which Y_0 can be reached in at most k steps in G. We will establish in a series of lemmas that with high probability, the cardinalities $|Y_0|, |Y_1|, |Y_2|, \dots$ grow at certain minimal rates in different ranges of k. Roughly, these are:

(i) Let k^* be minimal so that $|Y_{k^*}| \ge \frac{m}{2}$. Then the sets $Y_0, Y_1, \dots, Y_{k^*}$ grow by a factor of $\frac{11}{10}$. Since $|Y_0| \ge \varepsilon m/d$, this implies that $k^* = O(\log(1/\varepsilon))$.

(ii) Let ℓ^* be minimal so that $|Y_{k^*+\ell^*}| \ge m - \frac{4m}{e^4 d^3}$. The complement sets $[m] - Y_{k^*}, [m] - Y_{k^*+1}, \dots, [m] - Y_{k^*+\ell^*}$ shrink by a factor of $\frac{2}{3}$. This implies that $\ell^* = O(\log d)$.

(iii) Let L be maximal such that $Y_{k^*+\ell^*+L} \neq [m]$. The complement sets $[m] - Y_{k^*+\ell^*}, [m] - Y_{k^*+\ell^*+1}, \ldots, [m] - Y_{k^*+\ell^*+L}$ shrink by a factor of $d^{2/3}$.

These properties of G, which hold regardless of the decisions made by the algorithm that has stored S, are derived from expansion properties of the undirected graph G_u (see Remark 1), which in turn follow from h_1, h_2 being fully random. They are sufficient to show that the expected cost of finding a node in Y_0 by a BFS starting from two nodes $h_1(x)$ and $h_2(x)$ (where x is a new key) is $(1/\varepsilon)^{O(\log d)}$.

The full proofs of the very technical Lemmas 2 and 4 are postponed to Section 5.

Lemma 2. *Let $\varepsilon \le 0.1$ and $d \ge 15.8 \cdot \ln(\frac{1}{\varepsilon})$. Then there is a constant $\beta < 1$ such that with probability $1 - O(m\beta^m)$, each set of blocks Y that satisfies $\frac{\varepsilon}{1+\varepsilon}m \le r = |Y| \le \frac{1}{2}m$ is hit by at least $\frac{11}{10}rd$ keys from S.*

The *proof* can be found in Section 5.1.

We immediately set this lemma to use by showing that if Y_k is not too large then Y_{k+1} will be larger by a constant factor. This implies that already for some small k^*, the set Y_{k^*} covers half of $[m]$. To appreciate the relevance of this statement, we rephrase it: for at least half of the nodes in G a free node is "close by", namely, at most k^* steps away, where $k^* = O(\log(1/\varepsilon))$.

Lemma 3. *If the digraph G induced by S being stored meets the conclusion of Lemma 2, then there is some $k^* \le 1 + \log_{\frac{11}{10}}\left(\frac{(1+\varepsilon)}{2\varepsilon}\right) < 1 + 10\ln(1/\varepsilon)$ such that $|Y_{k^*}| > \frac{m}{2}$.*

Proof. Assume $k \ge 1$ and $|Y_{k-1}| \le \frac{1}{2}m$. Consider the set X_k of keys that hit Y_{k-1}. By the assumption, $|X_k| \ge \frac{11}{10}d|Y_{k-1}|$. For any key $x \in X_k$, there are two possibilities for the block $y = h_i(x)$ where x is stored ($i \in \{1, 2\}$): either $y \in Y_{k-1}$ ($\subseteq Y_k$) or $h_{2-i}(x) \in Y_{k-1}$ (then $y \in Y_k$ by the definition of Y_k). Thus, all keys $x \in X_k$ are stored in blocks in Y_k. Only $d|Y_{k-1}|$ of them can be stored in blocks in Y_{k-1}, so at least $\frac{1}{10}d|Y_{k-1}|$ of them must be stored in blocks in $Y_k - Y_{k-1}$, which implies that $|Y_k - Y_{k-1}| \ge \frac{1}{10}|Y_{k-1}|$; hence $|Y_k| \ge \frac{11}{10}|Y_{k-1}|$.

Now let k^* be minimal with $|Y_{k^*}| > \frac{1}{2}m$. Because there are exactly εn free cells in the table, we have $|Y_0| \ge \frac{\varepsilon n}{d} = \frac{\varepsilon}{1+\varepsilon}m$. Thus (by induction), $|Y_k| \ge \frac{\varepsilon}{1+\varepsilon}m\left(\frac{11}{10}\right)^k$ for $0 \le k < k^*$, whence we get $k^*-1 \le \left\lfloor \log_{\frac{11}{10}}\left(\frac{1+\varepsilon}{2\varepsilon}\right)\right\rfloor$. This proves the lemma. □

Next, we turn to those sets Y_k of a size larger than $\frac{m}{2}$, but bounded away from m by a constant factor. For these we show that the size $[m] - Y_k$ of the complement shrinks geometrically.

Lemma 4. *Let $d \ge 8$. Then there is some $\beta < 1$ such that with probability $1 - O(m\beta^m)$ we have that each set Y of blocks with $\frac{m}{2} \le |Y| \le m - \frac{4m}{e^4d^3}$ is hit by at least $n - \frac{9}{10}d(m - |Y|)$ keys.*

For the proof, see Section 5.2.

Using the last lemma, we see that in very few further steps in the sequence of Y_k's, all of $[m]$ excepting a small fraction (of $\frac{4}{e^4d^3}$) is covered. Turned the other way round, this means that not only half the nodes in G, but all of them excepting a small constant fraction are at a distance of only $O(\log(1/\varepsilon))$ to a free node.

Lemma 5. *If G meets the conclusions of Lemmas 2 and 4, and k^* satisfies $|Y_{k^*}| \ge \frac{m}{2}$, then there is some $\ell^* = O(\log d)$ such that $m - |Y_{k^*+\ell^*}| \le \frac{4m}{e^4d^3}$.*

Proof. Let ℓ^* be minimal such that $m - |Y_{k^*+\ell^*}| \le \frac{4m}{e^4d^3}$. If $\ell^* = 0$, there is nothing to show, so we assume $\ell^* > 0$. For $0 \le \ell \le \ell^*$ let $a_\ell = m - |Y_{k^*+\ell}|$. By Lemma 4, for $0 \le \ell < \ell^*$ the set $Y_{k^*+\ell}$ is hit by at least $n - \frac{9}{10}da_\ell$ keys. As in the proof of Lemma 3, one sees that all these keys must be stored in blocks in $Y_{k^*+\ell+1}$. Only $m - da_\ell$ of them can be stored in blocks in $Y_{k^*+\ell}$; hence at least $\frac{1}{10}da_\ell$ keys must be stored in blocks in $Y_{k^*+\ell+1} - Y_{k^*+\ell}$. This means that $|Y_{k^*+\ell+1} - Y_{k^*+\ell}| \ge \frac{1}{10}a_\ell$, and hence $a_{\ell+1} = m - |Y_{k^*+\ell+1}| \le \frac{9}{10}a_\ell$.

Since the a_ℓ's shrink geometrically by a factor of $\frac{9}{10}$, it is clear that $\ell^* \le \log_{\frac{10}{9}}(\frac{1}{2}/\frac{4}{e^4d^3}) = O(\log d)$. □

The following lemma states a standard expansion property of bipartite random graphs. (For a similar argument see [18, p. 109].)

Lemma 6. *Let $d \geq 20$, and $\gamma = \frac{4}{e^4 d^3}$. With probability $1 - O(m^{-d/2})$, we have that each set $X \subseteq S$ of keys with $d \leq |X| \leq \gamma dm$ hits more than $\Delta|X|$ different blocks, where $\Delta = \frac{1}{d^{1/3}} + \frac{1}{d}$.*

Proof. The probability that there is a set X of j keys, $d \leq j \leq \gamma dm$, that hits no more than Δj blocks, can be bounded by

$$\sum_{d \leq j \leq \gamma dm} \binom{n}{j} \binom{m}{\lfloor \Delta j \rfloor} \left(\frac{\lfloor \Delta j \rfloor}{m}\right)^{2j} \leq \sum_{d \leq j \leq \gamma dm} \left(\frac{en}{j}\right)^j \left(\frac{em}{\Delta j}\right)^{\Delta j} \left(\frac{\Delta j}{m}\right)^{2j}. \tag{37}$$

(We used the simple binomial bound (2).) Straightforward simplifications, using the fact that $d \geq 20$ implies that $\Delta < \frac{1}{2}$, lead to the bound $\sum_{d \leq j \leq \gamma dm} \left((e/2)^{3/2} \cdot d \cdot \sqrt{j/m}\right)^j$ for the right hand side in (37). For $j = d, d+1, \ldots, \lfloor \gamma dm \rfloor$, the terms in this sum are geometrically decreasing by a factor smaller than $\frac{1}{2}$, and hence the sum is bounded by $O(m^{-d/2})$. □

We conclude that for $k \geq k^* + \ell^*$ the complements of the sets Y_k shrink fast. We will see below that the sets $Y_0, \ldots, Y_{k^*+\ell^*}$ cause no problems in the time analysis, and hence we focus on the (standard) case that $Y_{k^*+\ell^*} \neq [m]$.

Lemma 7. *Assume that $m - |Y_{k^*+\ell^*}| \leq \gamma m$ and that the hash functions h_1, h_2 meet the conclusion of Lemma 6. Then the cardinalities $a_j = m - |Y_{k^*+\ell^*+j}|$, $j = 0, 1, 2, \ldots$, satisfy $a_j \leq d^{-2/3} \cdot a_{j-1}$ for $j = 1, 2, 3, \ldots$. Hence,*

$$m - |Y_{k^*+\ell^*+j}| \leq \gamma d^{-2j/3} m, \quad \text{for } j = 0, 1, 2, \ldots.$$

(In particular, there is some L with $Y_{k^*+\ell^*+L} \neq [m] = Y_{k^*+\ell^*+L+1}$.)

Proof. Fix $j \geq 1$. If $a_j = 0$, there is nothing to prove; thus assume $a_j \geq 1$. Then by the definitions, all a_j nodes in $[m] - Y_{k^*+\ell^*+j}$ are full. This means that the set E_j of edges in G with tails in $[m] - Y_{k^*+\ell^*+j}$ has cardinality da_j. By the assumption that the conclusion of Lemma 6 is satisfied, the edges in E_j touch at least $\Delta da_j = (d^{2/3}+1)a_j$ nodes overall. Only a_j of these nodes are in $[m]-Y_{k^*+\ell^*+j}$. By the definition of the sets Y_k, no edge in G can run from a node in $[m]-Y_{k^*+\ell^*+j}$ to a node in $Y_{k^*+\ell^*+(j-1)}$. Hence the heads of the edges in E_j hit at least $(d^{2/3}+1)a_j - a_j = d^{2/3}a_j$ distinct nodes in $[m] - Y_{k^*+\ell^*+(j-1)}$, which implies that $a_{j-1} \geq d^{2/3}a_j$. □

Lemma 8. *Assume $Y_0, Y_1, \ldots, Y_{k^*}, \ldots, Y_{k^*+\ell^*}, \ldots, Y_{k^*+\ell^*+L}$ are fixed and fulfill the expansion properties from Lemmas 3, 5 and 7. Assume further that $h_1(x), h_2(x)$ are random values in $[m]$. Then the expected number of edges probed in the BFS insertion procedure for x is $(1/\varepsilon)^{O(\log d)}$.*

Proof. Let N_x be the number of edges of G probed when x is inserted. Let $\sigma_k = \sum_{0<\kappa\leq k} d^\kappa < 2d^k$, for $k \geq 0$, and $k_x = \min\{k \mid h_1(x) \in Y_k \text{ or } h_2(x) \in Y_k\}$. Then the number of edges of G probed when inserting x is not larger than $2\sigma_{k_x}$. Thus, it is sufficient to estimate $\mathbf{E}(\sigma_{k_x})$. We have

$$\mathbf{E}(\sigma_{k_x}) = \sum_{q\geq 1} \mathbf{Prob}(\sigma_{k_x} \geq q) = \sum_{k\geq 1} \mathbf{Prob}(k_x \geq k) \cdot d^k. \tag{38}$$

The last sum in (38) is estimated in two pieces. We have

$$\sum_{1\leq k\leq k^*+\ell^*} \mathbf{Prob}(k_x \geq k) \cdot d^k \leq (k^* + \ell^*) d^{k^*+\ell^*}. \tag{39}$$

For the rest of the sum in (38), we notice that by Lemma 7

$$\begin{aligned}\sum_{k^*+\ell^*<k\leq k^*+\ell^*+L} \mathbf{Prob}(k_x \geq k) \cdot d^k &= d^{k^*+\ell^*} \cdot \sum_{1\leq j\leq L} \mathbf{Prob}(k_x \geq k^* + \ell^* + j) \cdot d^j \\ &\leq d^{k^*+\ell^*} \cdot \sum_{1\leq j\leq L} \mathbf{Prob}(h_1(x), h_2(x) \in [m] - Y_{k^*+\ell^*+(j-1)}) \cdot d^j \\ &\leq d^{k^*+\ell^*} \cdot \sum_{1\leq j\leq L} (d^{-2(j-1)/3})^2 \cdot d^j < 2d^{k^*+\ell^*+1}.\end{aligned} \tag{40}$$

The sum of the parts in (39) and (40) is bounded by $(k^* + \ell^* + 2)d^{k^*+\ell^*+1} = d^{O(\log(1/\varepsilon))} = (1/\varepsilon)^{O(\log d)}$. This shows that the expected number of edges probed in inserting x is bounded by $(1/\varepsilon)^{O(\log d)}$. □

To prove Theorem 2, we note that with probability $1 - O(m^{-d/2})$, the graph G satisfies the conclusions of Lemmas 2, 4 and 6. If this is the case, then the expansion properties of Lemmas 3, 5 and 7 hold, and we may apply Lemma 8 to obtain the claimed bound on the expected insertion time. If G does not have the expansion properties from Lemmas 3, 5 and 7, and Y_0 is reachable from $\{h_1(x), h_2(x)\}$, the BFS will find an augmenting path in time $O(n)$ — this gives a contribution of $O(m^{1-d/2})$ to the expected insertion time. In case Y_0 is not reachable from $\{h_1(x), h_2(x)\}$, the functions h_1, h_2 are not suitable for $S \cup \{x\}$, and we must perform a total rehashing for all these keys. By Theorem 1, this happens with probability $O(m^{1-d})$. It is easily seen that even if we simply insert the keys by the BFS procedure, and rehash again if necessary, the expected time for rebuilding the table is $O(n)$. Hence, this last case contributes $O(m^{2-d})$ to the expected insertion cost.

5. Proofs of two expansion lemmas

5.1. *Proof of Lemma 2*

Proof. A set of blocks Y, $|Y| = r$, is hit by $\frac{11}{10}rd$ keys with h_1 or h_2 if and only if at most $n - \frac{11}{10}rd$ keys hit $\overline{Y} = [m] - Y$ with both hash functions h_1 and h_2. By $F(r)$ we denote the probability that there exists a set Y of size r such that more than $n - \frac{11}{10}rd$ keys hit $\overline{Y}$ with both hash functions. To bound $F(r)$, we argue as in the proof of Lemma 1, using the Chernoff–Hoeffding bound (3). This yields

$$F(r) \le \binom{m}{r} \left(\frac{n\left(1-\frac{r}{m}\right)^2}{n - \frac{11}{10}rd}\right)^{n-\frac{11}{10}rd} \left(\frac{n - n\left(1-\frac{r}{m}\right)^2}{\frac{11}{10}rd}\right)^{\frac{11}{10}rd}. \tag{41}$$

We substitute $n = dm/(1+\varepsilon)$ and $r = \alpha m$, use (1), and simplify to get

$$F(r) \le \left[\frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}} \left(\frac{(1-\alpha)^2}{1-(1+\varepsilon)\frac{11}{10}\alpha}\right)^{\frac{d}{1+\varepsilon}-\frac{11}{10}\alpha d} \left(\frac{2-\alpha}{\frac{11}{10}(1+\varepsilon)}\right)^{\frac{11}{10}\alpha d}\right]^m. \tag{42}$$

To prove the lemma, it is sufficient to show that

$$\frac{1}{\alpha^\alpha(1-\alpha)^{1-\alpha}} \left(\frac{(1-\alpha)^2}{1-(1+\varepsilon)\frac{11}{10}\alpha}\right)^{\frac{d}{1+\varepsilon}-\frac{11}{10}\alpha d} \left(\frac{2-\alpha}{\frac{11}{10}(1+\varepsilon)}\right)^{\frac{11}{10}\alpha d} \tag{43}$$

is smaller than 1 for $\varepsilon/(1+\varepsilon) \le \alpha \le \frac{1}{2}$. This is the case if and only if d is at least

$$\frac{\alpha\ln\alpha + (1-\alpha)\ln(1-\alpha)}{\left(\frac{1}{1+\varepsilon} - \frac{11}{10}\alpha\right)\left(2\ln(1-\alpha) - \ln\left(1 - \frac{11(1+\varepsilon)\alpha}{10}\right)\right) + \frac{11\alpha}{10}\left(\ln(2-\alpha) - \ln\left(\frac{11(1+\varepsilon)}{10}\right)\right)} \tag{44}$$

for these α.

We wish to find an upper bound for the expression in (44). We call its numerator u and its denominator v. Both u and v are negative. If we can show that v'' is positive for $0 < \alpha \le \frac{1}{2}$, then v is concave, and bounded below by the secant $\tilde{v}$ through the points of v at $\alpha = 0$ and $\alpha = \frac{1}{2}$. Hence $u/\tilde{v} > u/v$.

The second derivative of v is

$$v'' = \frac{15\alpha + 176\varepsilon + 18 + 385\alpha\varepsilon - 89\alpha^2 + 132\alpha^2\varepsilon + 121\alpha^2\varepsilon^2 - 242\varepsilon^2 + 55\alpha^3(1+\varepsilon)}{5(2-\alpha)^2(1+\varepsilon)(10 - 11\alpha - 11\alpha\varepsilon)(1-\alpha)^2}. \tag{45}$$

The denominator of the latter term is positive if $\alpha < \frac{1}{2}$. We write the numerator as

$$(18 + \alpha(15 - 89\alpha + 55\alpha^2)) + (176 - 242\varepsilon + 385\alpha + 132\alpha^2 + 121\alpha^2\varepsilon + 55\alpha^3)\varepsilon, \tag{46}$$

a sum of two obviously positive terms (for $0 \le \alpha \le \frac{1}{2}$ and $0 \le \varepsilon \le 0.1$). Thus, v'' is positive.

The secant $\tilde{v}$ that intersects v at $\alpha = 0$ and $\alpha = \frac{1}{2}$ is

$$\tilde{v}(\alpha) = 2v(1/2)\alpha. \tag{47}$$

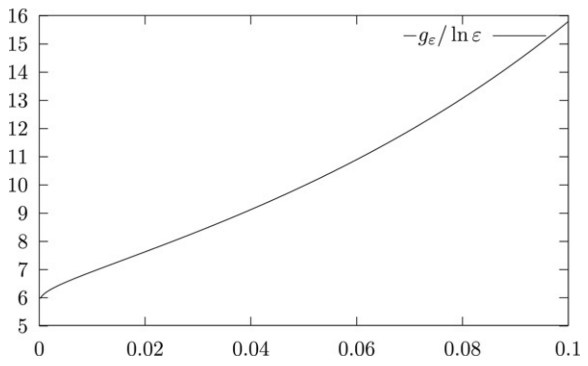


Fig. 4. The function $-g_\varepsilon/\ln\varepsilon$.

If we replace v by $\tilde{v}$ in (44), we get the upper bound

$$\frac{\alpha\ln\alpha+(1-\alpha)\ln(1-\alpha)}{2\alpha\left(\left(\frac{1}{1+\varepsilon}-\frac{11}{20}\right)\ln\left(\frac{5}{9-11\varepsilon}\right)+\frac{11}{20}\ln\left(\frac{15}{11(1+\varepsilon)}\right)\right)} \tag{48}$$

for (44). The expression

$$\left(\frac{1}{1+\varepsilon}-\frac{11}{20}\right)\ln\left(\frac{5}{9-11\varepsilon}\right)+\frac{11}{20}\ln\left(\frac{15}{11(1+\varepsilon)}\right) \tag{49}$$

is negative if $\varepsilon \le 0.1$. The expression

$$\frac{\alpha\ln(\alpha)+(1-\alpha)\ln(1-\alpha)}{\alpha} \tag{50}$$

is negative as well and has a positive derivative. So (48) is decreasing in α and is maximal at the left edge. Because $\alpha \ge \frac{\varepsilon}{1+\varepsilon}$, an upper bound for (48) is

$$\frac{\varepsilon\ln\varepsilon-(1+\varepsilon)\ln(1+\varepsilon)}{2\varepsilon\left(\left(\frac{1}{1+\varepsilon}-\frac{11}{20}\right)\ln\left(\frac{5}{9-11\varepsilon}\right)+\frac{11}{20}\ln\left(\frac{15}{11(1+\varepsilon)}\right)\right)} =: g_\varepsilon. \tag{51}$$

By choosing $d > g_\varepsilon$, we get $F(r) = O(\beta^m)$ for a constant $\beta < 1$.

Let us take a closer look at g_ε. We study the behaviour of $-g_\varepsilon/\ln\varepsilon$ for small ε (see Fig. 4). Obviously, $-g_\varepsilon/\ln\varepsilon$ is continuous. It is not hard to see that it is also strictly increasing for $\varepsilon \le 0.1$: Note that $-g_\varepsilon/\ln\varepsilon = A/(2B)$ where

$$A = 1-\frac{(1+\varepsilon)\ln(1+\varepsilon)}{\varepsilon\ln\varepsilon} \tag{52}$$

and

$$B = -\left(\frac{1}{1+\varepsilon}-\frac{11}{20}\right)\ln\left(\frac{5}{9-11\varepsilon}\right)-\frac{11}{20}\ln\left(\frac{15}{11(1+\varepsilon)}\right). \tag{53}$$

A is positive in $(0, 0.1]$ and strictly increasing in ε, which can be seen by examining its derivative. In the same way, one can check that B is a positive, strictly decreasing function. Therefore $A/(2B)$ is increasing. Furthermore, it is routine to check that

$$\lim_{\varepsilon\to 0}\frac{g_\varepsilon}{-\ln\varepsilon} = \frac{10}{7\ln 3-20\ln 5+11\ln 11} \approx 5.32. \tag{54}$$

For $\varepsilon = 0.1$ we get a value of $15.78\ldots$ for $-g_\varepsilon/\ln\varepsilon$. This finishes the proof of Lemma 2. □

Remark 5. For tighter bounds on ε, the constant 15.8 in Lemma 2 may be replaced by some smaller constant. For example, for $0 \le \varepsilon \le 0.01$, we have $g_\varepsilon < 7\ln(1/\varepsilon)$. If, for even smaller ε, one wishes to reduce the constant further, a modification in the proof of Lemma 5.1 is needed — instead of the expansion factor $\frac{11}{10}$, use some smaller constant

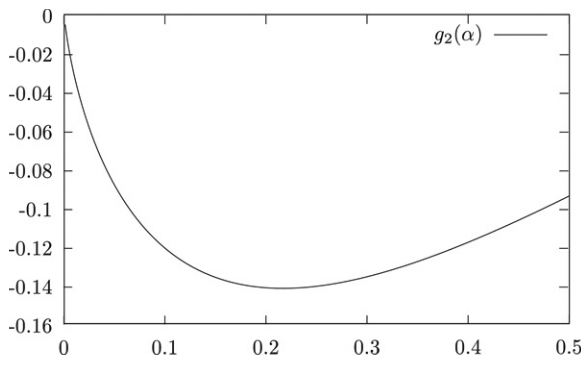


Fig. 5. The graph of g_2.

$1+\eta$, where $\eta > 0$. Since the calculations would be even more complex if we used η as another parameter, we chose to use $\eta = \frac{1}{10}$ somewhat arbitrarily.

5.2. *Proof of* Lemma 4

Proof. The set $Y \subseteq [m]$ is hit by at least $n - \frac{9d}{10}(m - |Y|)$ keys with h_1 or h_2 with one hash function if and only if at most $\frac{9d}{10}(m - |Y|) = \frac{9d}{10}|\overline{Y}|$ keys hit the set $\overline{Y}$ with *both* hash functions.

As before, we bound the probability $F_\varepsilon(r)$ that there is a set $\overline{Y}$ of size r, $\frac{4}{e^4 d^3} \le r \le \frac{m}{2}$, which is hit by more than $\frac{9d}{10}|\overline{Y}|$ keys with both hash functions by applying the Chernoff–Hoeffding-bound (3) and the binomial bound (1):

$$F_\varepsilon(r) \le \frac{m^m}{r^r(m-r)^{m-r}} \left(\frac{10r}{9(1+\varepsilon)m}\right)^{\frac{9}{10}rd} \left(\frac{10(m^2-r^2)}{(10m-9r(1+\varepsilon))m}\right)^{\frac{dm}{1+\varepsilon}-\frac{9}{10}rd}. \tag{55}$$

It is easy to see that the right hand side of (55) is decreasing in ε. Indeed, the derivative of $\ln(F_\varepsilon(r))$ with respect to ε is

$$-\frac{dm}{(1+\varepsilon)^2} \ln\left(\frac{10(m^2-r^2)}{10m^2 - 9(1+\varepsilon)rm}\right), \tag{56}$$

which is negative for $r \le m/2$, because $\frac{r}{m} \le \frac{1}{2} < \frac{9(1+\varepsilon)}{10}$, and hence

$$10r^2 < 9(1+\varepsilon)rm \Rightarrow 10(m^2-r^2) > 10m^2 - 9(1+\varepsilon)rm. \tag{57}$$

This leads to the following bound:

$$F_\varepsilon(r) < \frac{m^m}{r^r(m-r)^{m-r}} \left(\frac{10r}{9m}\right)^{\frac{9}{10}rd} \left(\frac{10(m^2-r^2)}{(10m-9r)m}\right)^{dm-\frac{9}{10}rd} =: g_1(r,d). \tag{58}$$

Next, we establish that $g_1(r,d)$ decreases if d increases: We substitute $r = \alpha m$, consider d as a continuous argument just for a moment, and examine

$$\frac{1}{m} \cdot \frac{\partial \ln g_1(r,d)}{\partial d} = \ln 10 + \frac{9}{10}\alpha \ln\left(\frac{\alpha}{9}\right) + \left(1 - \frac{9}{10}\alpha\right) \ln\left(\frac{1-\alpha^2}{10-9\alpha}\right) =: g_2(\alpha). \tag{59}$$

It is not very hard to see that $g_2(\alpha)$ is negative for $0 < \alpha < 1/2$ (see Fig. 5): Using a computer algebra system, one finds that

$$g_2''(\alpha) = \frac{(5\alpha^2 - 9\alpha + 5)(9\alpha^2 - 20\alpha + 9)}{5\alpha(10-9\alpha)(1-\alpha^2)^2}. \tag{60}$$

The numerator of g_2'' does not have any root in $(0, 1/2]$. From $g_2''(1/2) = 28/99 > 0$, it follows that $g_2''(\alpha) > 0$ for $0 < \alpha < 1/2$. Since g_2 is continuous in $(0, 1/2]$ and $\lim_{\alpha\to 0} g_2(\alpha) = 0$ and $g_2(1/2) < 0$, we conclude that $g_2(\alpha)$ is negative for $0 < \alpha < 1/2$. Hence $g_1(r,d)$ decreases if d increases.

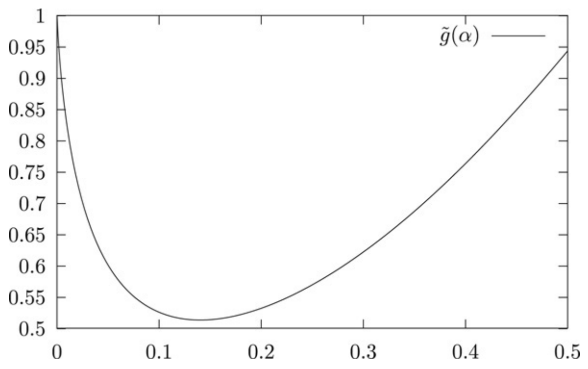


Fig. 6. The graph of $\tilde{g}$.

In particular, for $d \geq 8$ we may use the bound $g_1(r, d) \leq g_1(r, 8)$. With $r = \alpha m$, from (58), we get the following bound: $g_1(r, d) \leq \tilde{g}(\alpha)^m$, where

$$\tilde{g}(\alpha) = \frac{1}{\alpha^\alpha \cdot (1-\alpha)^{1-\alpha}} \cdot \left(\frac{10\alpha}{9}\right)^{7.2\alpha} \cdot \left(\frac{10(1-\alpha^2)}{10-9\alpha}\right)^{8-7.2\alpha}. \tag{61}$$

Again, we shortcut a tedious proof by calculus by looking at a plot of $\tilde{g}(\alpha)$ (see Fig. 6), which reveals that this function attains the value 1 for $\alpha = 0$ and has exactly one minimum point at about 0.14. This means that $\tilde{g}(\alpha)$ attains its maximum in $[\frac{4}{e^4 d^3}, \frac{1}{2}]$ at one of the border points, and hence $\tilde{g}(\alpha) \leq \max\{\tilde{g}(\frac{4}{e^4 d^3}), \tilde{g}(\frac{1}{2})\} < 1$, for all $d \geq 8$, $\frac{4}{e^4 d^3} \leq \alpha \leq \frac{1}{2}$. By numerical evaluation, the maximum is seen to be $\tilde{g}(\frac{4}{e^4 d^3})$. This implies that $F_\varepsilon(r) \leq \tilde{g}(\frac{4}{e^4 d^3})^m$ for all r in $[\frac{4m}{e^4 d^3}, m/2]$, and hence

$$\sum_{\frac{4m}{e^4 d^3} \leq r \leq m/2} F_\varepsilon(r) < m \cdot \beta^m,$$

for some constant $\beta < 1$ (which of course depends on d). □

6. Conclusion

We obtained new results for a natural data allocation problem arising in different contexts: balanced allocation with two choices, edge orientation in random graphs, and dynamic dictionaries with worst case constant access time. We showed that a slack factor (fraction of unused positions) of $\varepsilon > 0$ can be achieved with maximum bucket size $d = O(\log(1/\varepsilon))$, both in the static and in the dynamic case.

It is an intriguing open problem to analyze at least one variant of the random-walk insertion procedure, if possible establishing a bound of $O(\ln(1/\varepsilon))$ on the expected number of blocks probed (while maintaining the bound $d = O(\log(1/\varepsilon))$).

Acknowledgements

We thank the anonymous referees for comments helpful in improving the presentation of the material.

Appendix. Experiments

In two experiments, we compared the performance of four methods for implementing a dynamic dictionary:

- blocked cuckoo hashing (*cuckoo-block*), as described in the paper;
- a cuckoo-linear-probing scheme (*cuckoo-lp*), a variant in which a key x is assigned to two positions $h_1(x)$ and $h_2(x)$ in a table of size $m = (1+\varepsilon)n$ and may be placed in one of the cells $(h_j(x) + r) \bmod m$, for $j = 1, 2$ and $0 \leq r < d$;
- linear probing (*lp*) with one hash function;
- cuckoo hashing with d functions (*cuckoo-d*).

ε	cuckoo-d		cuckoo-block		cuckoo-lp		lp
	time	d	time	d	time	d	time
0.5	1.618	3	0.924	5	0.908	5	0.38
0.2	2.142	3	0.938	8	0.94	5	0.388
0.1	2.678	4	0.952	16	0.98	8	0.4
0.05	3.376	5	0.966	16	1.024	11	0.418
0.02	4.46	6	1.002	32	1.094	15	0.466
0.01	5.354	7	1.02	32	1.152	18	0.546

Fig. 7. Average insertion time in μs and the "best" d for a key when filling an empty array.

ε	unsuccessful search						
	cuckoo-d		cuckoo-block		cuckoo-lp		lp
	time	d	time	d	time	d	time
0.5	1.286	3	0.814	2	0.816	2	0.378
0.2	1.278	3	0.82	3	0.806	2	0.43
0.1	1.272	3	0.82	3	0.814	3	0.548
0.05	1.678	4	0.826	4	0.81	3	0.934
0.02	2.086	5	0.834	5	0.81	3	3.012
0.01	2.092	5	0.834	6	0.808	3	10.114

ε	successful search						
	cuckoo-d		cuckoo-block		cuckoo-lp		lp
	time	d	time	d	time	d	time
0.5	0.846	3	0.708	2	0.794	2	0.354
0.2	0.838	3	0.76	3	0.788	2	0.36
0.1	0.834	3	0.766	3	0.806	3	0.368
0.05	1.048	4	0.79	4	0.814	5	0.384
0.02	1.24	5	0.802	5	0.82	6	0.416
0.01	1.24	5	0.81	6	0.82	6	0.462

Fig. 8. Average lookup time for a search in μs and the "best" d.

In the first experiment, $n = 10^6$ distinct keys were inserted into an empty table of size $(1 + \varepsilon)n$. Then 10^6 keys, all different from the first ones, were searched. Thus the time for building a dictionary and the time for a negative lookup were measured.

The keys were randomly chosen from $[2^{28}]$; the hash functions were random polynomials from $\{h\colon x \mapsto (\sum_{i=0}^{3} a_i x^i) \bmod p \bmod m \mid a_0, \ldots, a_3 \in [2^{28}]\}$, where p is a large prime number and m is the number of blocks for *cuckoo-block* and the size of the array otherwise.

Insertion for all cuckoo hashing variants was implemented in the random-walk fashion, see Section 2.2. The experiments[3] were executed for different values of ε and d. For each of the cuckoo hashing variants, each task, and for each ε we noted the "best" d, which minimized the measured time for the respective operation. Further, the time needed by linear probing was measured (see tables in Figs. 7 and 8).

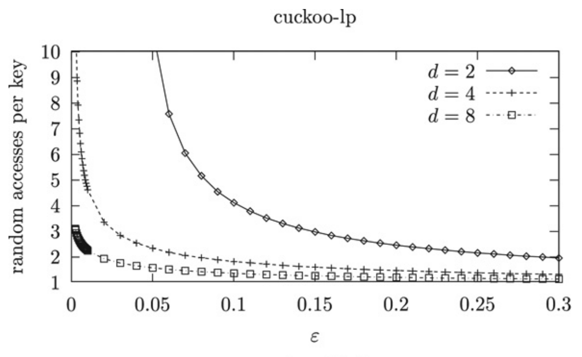
Looking at the tables in Figs. 7 and 8 one notes that decreasing ε forces larger d to be chosen, but affects the insertion time and the negative lookup time of *cuckoo-block* and *cuckoo-lp* only a little. On the other hand, *cuckoo-d* is affected more: each additional hash function needed to make up for a smaller ε increases the cost of a negative

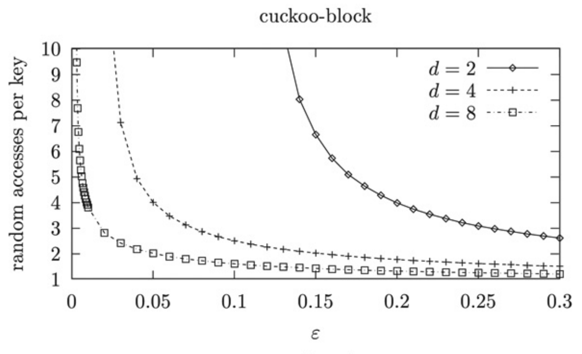
[3] Processor: Intel(R) Pentium(R) 4 CPU 2.40 GHz stepping 07 (8 K L1 cache, 512 K L2 cache), board: GA-8PE800 i845PE ATX, RAM: DDR-333 512 MB, environment: SuSE Linux 9.1 (kernel 2.6.5), compiler: Intel C Compiler for Linux, version 8.0, optimization options: -O3 -xN -ipo.

d	cuckoo-lp	cuckoo-block	A	B
2	0.038394	0.115584	0.535156	0.7358
3	0.007117	0.043228	0.208261	0.5413
4	0.001975	0.02061	0.105351	0.3983
5	0.000724	0.01102	0.059569	0.2931
6	0.000332	0.006375	0.035834	0.2156
7	0.000178	0.003828	0.022396	0.1586
8	0.000113	0.002393	0.014370	0.1167
9	0.000076	0.001551	0.009402	0.0859
10	0.000062	0.001024	0.006234	0.0632
11	0.000048	0.000686	0.004176	0.0465

Columns labeled with A denote the smallest ε satisfiying relation (20) and columns labeled with B denote the smallest ε satisfying the conclusion of Lemma 1.

Fig. 9. Smallest ε for given d; $n \approx 2 \times 10^7$.





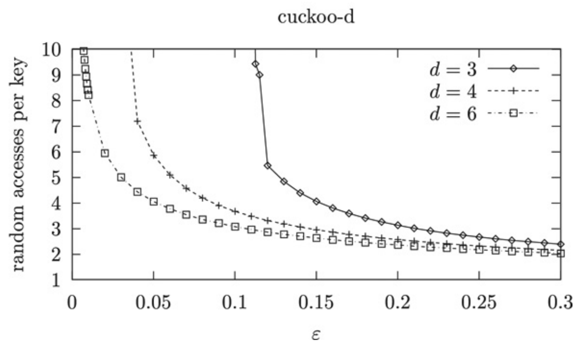


Fig. 10. Average number of random accesses per key.

lookup operation by about 0.4 µs. Moreover, we observe a dramatic increase in the negative lookup time for *lp* if ε goes below 0.02. For positive lookups, when averaging over keys that are present in the table, linear probing cannot be beaten for sets of the size considered here; but one should bear in mind that the worst-case lookup time is as slow as an unsuccessful search.

In a second experiment, randomly chosen keys were inserted into an empty table with 2×10^7 cells as long as possible for the cuckoo hashing variants with two hash functions. An overall time bound Kn for some constant $K = K(\varepsilon)$ was fixed beforehand. This gives estimates on the maximal possible space utilization for a given d (see table in Fig. 9). The experiment shows that *cuckoo-lp* packs the keys more tightly than *cuckoo-block*, at least in the static case and for sets of the size considered here. For comparison, we have listed the bounds on ε for a given d that result from numerically optimizing with bound (20) and from applying Theorem 1.

To throw some light on the reasons for the time needed for the operations, we include and briefly comment on results of work done by a student [20] that deals with the number of random memory accesses while inserting 10^6 randomly chosen distinct keys into an empty array (see Fig. 10). By “random memory access”, we mean the access to a block of d contiguous memory cells for *cuckoo-lp* and *cuckoo-block*, while for *cuckoo-d* each memory access is a random one. Note that usually each random memory access will cause a cache fault; therefore it is a good indicator for the runtime.

The main observation from Figs. 9 and 10 is that if for a given d the value of ε is large enough, that is ε is in the range of column B of the table in Fig. 9, then the average number of random accesses for *cuckoo-lp* and *cuckoo-block* is quite small (below 3). This is no surprise, since for most of the keys that are inserted at an early stage only one random memory access is needed to find a free cell. When ε gets close to the limiting values as shown in the table in Fig. 9 (columns “cuckoo-lp” and “cuckoo-block”), the number of random accesses increases dramatically. For a fixed d and a fixed ε, the number of random accesses for *cuckoo-lp* is noticeably smaller than for *cuckoo-block*. For completeness, we also give the results for *cuckoo-d*. (Note that in *cuckoo-d*, one key has d cells to go to while in *cuckoo-block* with block size d, it has $2d$ cells to go to, so a direct comparison between these two situations is impossible.)

References

[1] Y. Azar, A.Z. Broder, A.R. Karlin, E. Upfal, Balanced allocations, SIAM J. Comput. 29 (2000) 180–200.

[2] P. Berenbrink, A. Czumaj, A. Steger, B. Vöcking, Balanced allocations: The heavily loaded case, in: 32nd STOC, ACM, 2000, pp. 745–754.

[3] B. Bollobas, Random Graphs, 2nd edition, Cambridge University Press, 2001.

[4] A. Czumaj, F. Meyer auf der Heide, V. Stemann, Contention resolution in hashing based shared memory simulations, SIAM J. Comput. 29 (2000) 1703–1739.

[5] A. Czumaj, Ch. Riley, Ch. Scheideler, Perfectly balanced allocation, in: RANDOM-APPROX, in: LNCS, vol. 2764, Springer, 2003, pp. 240–251.

[6] R. Diestel, Graph Theory, Springer, New York, 1997.

[7] M. Dietzfelbinger, Applications of a splitting trick, 2007, in preparation.

[8] M. Dietzfelbinger, F. Meyer auf der Heide, Dynamic hashing in real time, in: J. Buchmann, et al. (Eds.), Informatik-Festschrift zum 60. Geburtstag von Günter Hotz, B. G. Teubner, 1992, pp. 95–119.

[9] M. Dietzfelbinger, P. Woelfel, Almost random graphs with simple hash functions, in: 35th STOC, ACM, 2003, pp. 629–638.

[10] D. Fotakis, R. Pagh, P. Sanders, P. Spirakis, Space efficient hash tables with worst case constant access time, Theory of Computing Systems 38 (2005) 229–248.

[11] A. Frieze, Personal communication in [1], 1990.

[12] M. Frigo, C.E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, in: 40th FOCS, IEEE, 1999, pp. 285–298.

[13] T. Hagerup, Ch. Rüb, A guided tour of Chernoff bounds, Inform. Process. Lett. 33 (1990) 305–308.

[14] R. Karp, Random graphs, random walks, differential equations and the probabilistic analysis of algorithms, in: 15th STACS, in: LNCS, vol. 1373, Springer, 1998, pp. 1–2.

[15] R. Karp, M. Luby, F. Meyer auf der Heide, Efficient PRAM simulations on a distributed memory machine, Algorithmica 16 (1996) 517–542.

[16] M. Mitzenmacher, A.W. Richa, R. Sitaraman, The power of two random choices: A survey of techniques and results, in: S. Rajasekaran, et al. (Eds.), in: Handbook of Randomized Computing, vol. 1, Kluwer Academic Press, 2001, pp. 255–312.

[17] R. Motwani, Average-case analysis of algorithms for matchings and related problems, J. ACM 41 (6) (1994) 1329–1356.

[18] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.

[19] A. Östlin, R. Pagh, Uniform hashing in constant time and linear space, in: 35th STOC, ACM, 2003, pp. 622–628.

[20] A. Ott, Experimenteller Vergleich einiger neuer Verfahren zur Konstruktion von dynamischen Wörterbüchern mit fixer Zugriffszeit (in German), Studienarbeit, Technische Universität Ilmenau, Fakultät IA, 2004.

[21] R. Pagh, F.F. Rodler, Cuckoo hashing, J. Algorithms 51 (2004) 122–144.

[22] R. Panigrahy, Efficient hashing with lookups in two memory accesses, in: 16th SODA, ACM-SIAM, 2005, pp. 830–839.

[23] P. Sanders, Algorithms for scalable storage servers, in: SOFSEM 2004, in: LNCS, vol. 2932, Springer, 2004, pp. 82–101.
[24] P. Sanders, Fast priority queues for cached memory, in: 1st Workshop ALENEX, in: LNCS, vol. 1619, Springer, 1999, pp. 312–327.
[25] P. Sanders, Reconciling simplicity and realism in parallel disk models, in: Parallel Data Intensive Algorithms and Applications, Parallel Computing 28 (5) (2002) 705–723 (special issue).
[26] P. Sanders, S. Egner, J. Korst, Fast concurrent access to parallel disks, Algorithmica 35 (1) (2003) 21–55.
[27] A. Siegel, On universal classes of extremely random constant-time hash functions, SIAM J. Comput. 33 (2004) 505–543.