# Cache-Oblivious Hashing[*]

Rasmus Pagh — IT University of Copenhagen, Copenhagen, Denmark, pagh@itu.dk

Zhewei Wei, Ke Yi, Qin Zhang — Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong, China, {wzxac, yike, qinzhang}@cse.ust.hk

**Abstract**

The hash table, especially its external memory version, is one of the most important index structures in large databases. Assuming a truly random hash function, it is known that in a standard external hash table with block size $b$, searching for a particular key only takes expected average $t_q = 1 + 1/2^{\Omega(b)}$ disk accesses for any load factor $\alpha$ bounded away from 1. However, such near-perfect performance is achieved only when $b$ is known and the hash table is particularly tuned for working with such a blocking. In this paper we study if it is possible to build a cache-oblivious hash table that works well with any blocking. Such a hash table will automatically perform well across all levels of the memory hierarchy and does not need any hardware-specific tuning, an important feature in autonomous databases.

We first show that linear probing, a classical collision resolution strategy for hash tables, can be easily made cache-oblivious but it only achieves $t_q = 1 + \Theta(\alpha/b)$ even if a truly random hash function is used. Then we demonstrate that the block probing algorithm [20] achieves $t_q = 1 + 1/2^{\Omega(b)}$, thus matching the cache-aware bound, if the following two conditions hold: (a) $b$ is a power of 2; and (b) every block starts at a memory address divisible by $b$. Note that the two conditions hold on a real machine, although they are not stated in the cache-oblivious model. Interestingly, we also show that neither condition is dispensable: if either of them is removed, the best obtainable bound is $t_q = 1 + O(\alpha/b)$, which is exactly what linear probing achieves.

## 1 Introduction

The hash table is one of the most fundamental index structures in databases. It stores a set of $n$ keys from a universe $[u]$ in linear space, while allowing us to search for any particular key efficiently. It is also one of the simplest data structures: Let $h : [u] \to [r]$ be a hash function. The table has size $r \geq n$ and we simply store key $x$ in position $h(x)$. If that position already contains some other key, one can use various collision resolution strategies, among which *chaining* and *linear probing* are the most common ones. In chaining, we simply store all keys that are mapped to the same position in a list associated with that position. In linear probing, if position $h(x)$ is already occupied when $x$ is being inserted, we successively probe positions $h(x), h(x)+1, \ldots, r-1, 0, 1, \ldots, h(x)-1$ until an empty position is found and we will put $x$ there. To perform a search on $x$, we follow the same probing sequence, until $x$ is found or an empty position is encountered, in which case we know that $x$ is not stored in the table. It is known that linear probing generally outperforms chaining in practice due to its sequential access pattern, provided that the *load factor* $\alpha = n/r$ is not too close to 1.

[*]A preliminary version of this paper was presented at the ACM Symposium on Principles of Database Systems, 2010.

The mathematical analysis of hashing is usually considered as the birth of analysis of algorithms [14], and it is still attracting a lot of attention nowadays. Most analyses on hashing assume $h$ to be a truly random function, i.e., each $h(x)$ is independently uniformly distributed on $[r]$. Under such an assumption, Knuth [14] showed that the expected average number of probes during a search using linear probing is (averaged over all keys):

$$C_n \approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right) \qquad \text{(successful search);}$$
$$C'_n \approx \frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right) \qquad \text{(unsuccessful search).}$$

Thus, for a typical load factor $\alpha = 0.7$, we expect to make $2.17$ probes if the searched key is in the table, and $6.05$ probes if it is not.

In large databases, the hash table is usually stored in external memory, and data is accessed in terms of blocks. In this setting, we care about the number of blocks accessed (I/Os) when performing a search. The number of I/Os is clearly at most the number of probes, but such a naive analysis is too pessimistic. Interestingly, Knuth [14] showed that the external version of linear probing has a search cost of $1 + 1/2^{\Omega(b)}$ I/Os (for both successful and unsuccessful searches), where $b$ is the block size. Here and further we assume that the load factor $\alpha$ is bounded away from 1. In the external version of linear probing, the table consists of $r/b$ blocks, and correspondingly we use a hash function $h : [u] \to [r/b]$. To do a search on $x$, we successively access blocks $h(x), h(x) + 1, \ldots$ until $x$ is found or a non-full block is encountered. The intuitive explanation for this extremely close-to-one I/O cost is that since a block has size $b$, we will not have a collision unless more than $b$ keys are hashed into this block, which happens with probability exponentially small in $b$. Knuth [14] actually derived the constant in the big-Omega, showing that for reasonably large $b$ (larger than 10), the number of I/Os is very close to 1, much smaller than the number of probes. Meanwhile, a natural external version of chaining also achieves the same bound. These results basically have explained why hash tables work so well in external memory.

These classical analyses assumed a simple two-level memory model [2], where the (sufficiently large) external memory is partitioned into blocks of size $b$ and are fetched into the internal memory of size $m$ as they are probed. Here both sizes are measured in terms of $(\log u)$-bit words. Starting in the late 90's tremendous efforts have been devoted to the design and analysis of data structures that work well not only in a two-level memory model, but also in a memory hierarchy that consists of any number of levels, where each level has a different capacity $m$ and block size $b$. Among them, the most successful approach is the *cache-oblivious* model [10] due to its elegance and simplicity. This model actually only features two levels of memory: a data structure is laid out in external memory and accessed in exactly the same way as in the standard two-level model, but the additional requirement is that the structure is unaware of the block size $b$, or equivalently, the structure is laid out in external memory in a way that works for all block sizes[1]. Thus a cache-oblivious data structure automatically works in a memory hierarchy. More precisely, if we can show that the cost of some operation on a cache-oblivious structure is $f(n, b)$ I/Os in the two-level model, then the number of block transfers will always be $f(n, b)$ between any two levels in a memory hierarchy with multiple levels, where the $b$ simply becomes the block size of that corresponding level. Another major benefit of cache-oblivious algorithms and data structures is that they achieve their guaranteed performance without any hardware-specific tuning. This is particularly important in autonomous databases, and is in fact the main motivation of the recent efforts in bringing cache-oblivious techniques to databases, such as EaseDB [12].

[1]Strictly speaking the structure should be unaware of both $m$ and $b$. But for most data structure problems the operations on the structure are always oblivious to $m$, so we only need to require that the layout works for all $b$.

Note that the external versions of linear probing and chaining mentioned above only work for a single $b$, so they are not cache-oblivious. In this paper, we investigate whether it is possible to lay out a hash table such that its search cost matches its cache-aware version, i.e., $1 + 1/2^{\Omega(b)}$ I/Os, for all block sizes $b$.

**Our results** A straightforward way of making the hash table cache-oblivious is to simply use linear probing but ignoring the blocking altogether[2]. One would expect it to work well irrespective of the block size since it uses only sequential probes. However, in Section 3 we show that its search cost is $1 + O(\alpha/b)$ I/Os even assuming a truly random hash function. In fact, we also derive the constant in the big-Oh, which depends on $C_n$ and $C'_n$. This result was first stated by Qi and Martel [22], but we give a complete proof in Section 3. This is worse than its cache-aware version that is particularly tuned to work with a single $b$. The gap is in some sense exponential, if we are concerned with the fraction of keys that cannot be found with a single I/O (note that an average search cost of $t_q = 1 + \varepsilon$ means that at most a fraction of $\varepsilon$ keys need two or more I/Os).

Next, we explore other collision resolution strategies to see if they work better in the cache-oblivious model. In Section 4, we show that the *blocked probing* algorithm [20] achieves the desired $1+1/2^{\Omega(b)}$ search cost, but under the following two conditions: (a) $b$ is a power of 2; and (b) every block starts at a memory address divisible by $b$. In addition, we have analyzed the performance of blocked probing when the hash function has limited independence: We show that with a $k$-wise independent hash function, the expected I/O cost of a search is $1 + O\left(\left(\frac{k-1}{e^{2/3}(1-\alpha)^2 b}\right)^{(k-1)/2}\right)$. Since a $k$-wise independent hash function is also $k'$-wise independent for all $k' \le k$, as long as $k \ge (1-\alpha)^2 b + 1$, the bound becomes $1 + 2^{-\Omega((1-\alpha)^2 b)} = 1 + 2^{-\Omega(b)}$, matching that of a truly random hash function.

Note that neither of the two conditions above is stated in the cache-oblivious model, but they indeed hold on all real machines. This raises the theoretical question of whether $1 + 1/2^{\Omega(b)}$ is achievable in the "true" cache-oblivious model. In Section 5, we show that neither condition is dispensable. Specifically, we prove that if the hash table is only required to work for a single $b$ but an arbitrary shift of the layout, or if (b) holds but the hash table is required to work for all $b$, then the best obtainable search cost is $1 + O(\alpha/b)$ I/Os, which exactly matches what linear probing achieves. Our lower bound model allows a truly random hash function to be used and puts no restrictions on the structure of the hash table, except that each key is treated as an atomic element, known as the *indivisibility* assumption.

**Related results** Hashing is perhaps one of the most studied problems in computer science. Most work on hashing assumes a truly random function. Since such a function requires a large space to describe, there are also a lot of work on hashing using explicit and efficient hash functions [6, 20]. Meanwhile, although most work focuses on the expected search cost, there are also hashing schemes that guarantee good worst-case search costs [9, 21]. Hashing has been well studied in the external memory model. The $1 + 1/2^{\Omega(b)}$ search cost holds as long as the load factor $\alpha$ is bounded away from 1 [14], and there are various techniques in the database literature to keep the load factor in a desired range, such as *extensible hashing* [8] or *linear hashing* [17]. Jensen and Pagh [13] designed a hashing scheme that has $\alpha = 1 - O(1/\sqrt{b})$ while supporting searches with $1 + O(1/\sqrt{b})$ I/Os. In all these hashing schemes a small faction of the keys still need two or more disk accesses to retrieve. There are other schemes that guarantee a single I/O to retrieve any key [11, 16], but they all need the internal memory to have size $m = \Theta(n/b)$. Note that those hashing schemes achieving $t_q = 1 + \varepsilon$ only need the internal memory to store a constant number of blocks.

[2]Chaining would perform worse cache-obliviously because the list associated with each position is not laid out consecutively.

The cache-oblivious model was proposed by Frigo et al. [10], which introduces a clean and elegant way to modelling memory hierarchies. Previous approaches attempted to model a memory hierarchy directly, but did not have much success due to the complicated models. Since then, cache-oblivious algorithms and data structures have received a lot of attention, and most fundamental problems have been solved. For example, cache-oblivious sorting takes $O(\frac{n}{b}\log_{m/b}\frac{n}{b})$ I/Os [10], and a cache-oblivious B-tree takes $O(\log_b n)$ I/Os for a search [4]. Please see the survey [7] for other results. In most cases, the cache-oblivious bounds match their cache-aware versions, and it has always be an interesting problem to see for what problems do we have a separation between the cache-oblivious model and the cache-aware model. Until today there have been only three separation results [1, 3, 5]; our lower bound adds to that list, furthering our understanding of cache-obliviousness.

## 2 Preliminaries

Let $[x] \overset{\text{def}}{=} \{0, 1, \dots, x-1\}$. Throughout this paper $S$ denotes a subset of the universe $U = [u]$, and $h$ will denote a hash function from $U$ to $R \overset{\text{def}}{=} [r]$. We denote the elements of $S$ by $\{x_1, x_2, \dots, x_n\}$, and refer to the elements of $S$ as *keys*. We let $n \overset{\text{def}}{=} |S|$ and $\alpha \overset{\text{def}}{=} n/r$.

The classic results assume that the hash function $h$ distributes each key $x$ independently uniformly on $R$. Such a function is called a truly random function. This assumption is unrealistic, since to simply store a truly random hash function requires $u \log r$ bits. To bridge the gap between hashing algorithms and their analysis, Carter and Wegman introduced *universal hashing* [6]. A family $\mathcal{H}$ of functions from $U$ to $R$ is *k-wise independent* if for any $k$ distinct elements $x_1, \dots, x_k \in U$ and $h$ chosen uniformly at random from $\mathcal{H}$, the random variables $h(x_1), \dots, h(x_k)$ are independent. We refer to the variable

$$\bar{\alpha} \overset{\text{def}}{=} n \max_{x\in U, \rho \in R} \Pr_{h\in\mathcal{H}}[h(x) = \rho]$$

as the *maximum load* of $\mathcal{H}$. If $\mathcal{H}$ distributes hash function values of all elements of $U$ uniformly on $R$, we will have $\bar{\alpha} = \alpha$, and in general $\bar{\alpha} \geq \alpha$. We assume that all families used in this paper are uniform so we do not distinguish $\bar{\alpha}$ from $\alpha$. For non-uniform families, all results in this paper hold if we substitute $\alpha$ with $\bar{\alpha}$.

Carter and Wegman [26] exhibited the following family of $k$-wise independent hash functions where $U = [p]$, $R = [r]$, and $p$ is a prime:

$$\mathcal{H}_k = \left\{ h : \, h(x) = \Big( (a_{k-1}x^{k-1} + \cdots + a_0) \bmod p \Big) \bmod r, \, a_j \in [p] \right\}.$$

This could be easily verified: observe that the family of degree $k-1$ polynomials in the finite field $Z_p$ is $k$-wise independent; to obtain a smaller range $R = [r]$ we may map integers in $[p]$ down to $[r]$ by a modulo $r$ operation. This operation preserves independence, only making the family (slightly) non-uniform. Specifically, the maximum load $\bar{\alpha}$ for this family is in the range $[\alpha, (1 + r/p)\alpha]$. By choosing $p$ much larger than $r$ we can make $\bar{\alpha}$ arbitrarily close to $\alpha$.

## 3 Analysis of Linear Probing in the Cache-Oblivious Model

Linear probing while ignoring the blocking is naturally cache-oblivious. In this section we analyze its search I/O cost, which turns out to delicately depend on $C_n$ and $C'_n$, the expected average number of probes in a successful and unsuccessful search, respectively. Note that the equalities in the theorem below are exact, though we only know the asymptotic formulas for $C_n$ and $C'_n$.

**Theorem 1** *Suppose the linear probing algorithm uses a truly random hash function $h$. Let $CO_n$ and $CO'_n$ denote the expected average number of I/Os for a successful and an unsuccessful search, respectively. For any block size $b$, we have*

$$CO_n = 1 + (C_n - 1)/b;$$
$$CO'_n = 1 + (C'_n - 1)/b.$$

*Proof*: Let $r$ be the size of the hash table, which is divided into $r/b$ blocks $B_0, \ldots, B_{r/b-1}$ (assuming that $r$ is a multiple of $b$ for simplicity). The block $B_l$ spans positions $lb, lb+1, \ldots, lb+b-1$. Consider an unsuccessful search for a key $x$. Define $p(i,j)$, $i \neq j$, to be the event that the hash table has positions $i$ through $j$ occupied (wrapping around when necessary). Note that the number of occupied positions is $n$, so $p(i,j) = 0$ for any $j \notin \{i, i+1, \ldots, i+n-1\}$ (wrapping around when necessary). By the circular symmetry of linear probing and the uniform hash function assumption, $p(0,k)$ is exactly the probability that an unsuccessful search for a key $x$ takes at least $k+2$ probes. Thus we have:

$$C'_n = 1 + \sum_{k=0}^{n-1} p(0,k). \tag{1}$$

Let $p_k$ be the probability that an unsuccessful search takes at least $k+1$ I/Os. Below we will relate $p_k$ with the $p(0,k)$'s. By the uniformity of the hash function $h$, we assume that $h(x)$ lies in the first block. Note that for a insertion to cost at least $k+1$ I/Os, positions $h(x)$ through $kb-1$ must be occupied. Since $h(x)$ hits position 0 through $b-1$ with same probability, we have

$$\begin{aligned} p_k &= \frac{1}{b}\sum_{i=0}^{b-1} p(i, kb-1) \\ &= \frac{1}{b}\sum_{i=0}^{b-1} p(0, kb-i-1) \qquad \text{(Since } h \text{ is a truly random function.)} \end{aligned}$$

Now we can compute $CO'_n$ as follows:

$$\begin{aligned} CO'_n &= 1 + \sum_{k=1}^{n/b} p_k \\ &= 1 + \sum_{k=1}^{n/b} \frac{1}{b} \sum_{i=0}^{b-1} p(0, kb-i-1) \\ &= 1 + \frac{1}{b}\sum_{j=0}^{n-1} p(0,j). \end{aligned}$$

Plugging in (1) to the equation and we have

$$CO'_n = 1 - \frac{1}{b} + \frac{1}{b}\left(1 + \sum_{j=0}^{n-1} p(0,j)\right)$$

$$= \; 1 - \frac{1}{b} + \frac{C'_n}{b}.$$

For the successful query cost $CO_n$, we relate it with $CO'_k$ and $C'_k$, the expected average number of I/Os and probes respectively of an unsuccessful search on a table of size $k$ for $k = 0, \dots, n-1$, using the same transformation in [14]:

$$CO_n = \frac{1}{n} \sum_{k=0}^{n-1} CO'_k = 1 - \frac{1}{b} + \frac{\sum_{k=0}^{n-1} C'_k}{nb} = 1 - \frac{1}{b} + \frac{C_n}{b}.$$

□

Combing with Knuth's result that $C_n \approx \frac{1}{2}(1 + \frac{1}{1-\alpha})$ and $C'_n \approx \frac{1}{2}(1 + (\frac{1}{1-\alpha})^2)$, we conclude that the I/O cost of directly applying linear probing in the cache-oblivious model is $1 + \Theta(\alpha/b)$, which is a lot worse than its the external version that is aware of the blocking.

## 4 Blocked Probing

Standard linear probing maintains the invariant that each key $x$ is placed as close as possible to position $h(x)$ in the probe sequence. *Blocked probing* is a variant of linear probing proposed by Pagh et al. [20], who used it to derive optimal performance (as a function of $\alpha$) assuming only 5-wise independent hash functions. In this section, we demonstrate that blocked probing also achieves the desired $1 + 2^{-\Omega(b)}$ I/O bound in the cache-oblivious model, under the assumptions that the block size $b$ is a power of 2 and the memory blocks are $b$-aligned.

### 4.1 Algorithm description

Let $[r] = \{0, 1, \dots r-1\}$ denote the hash table, where $r$ is a power of two. It is also assumed that $r$ is fixed, i.e., there is no notion of dynamically adjusting the capacity of the hash table; at the end of this section we sketch how to handle the general case. Suppose that the key $x$ is stored in location $i_x$, we define the distance measure $d(x, i_x)$ to be the position of the most significant bit in which $h(x)$ and $i_x$ differ (the least significant bit is said to be at position 1), and $d(x, i_x) = 0$ in case $i_x = h(x)$. Let $I(x, j) = \{i \mid d(x, i) \le j\}$. Note that $I(x, j)$ is the *aligned* block of size $2^j$ that contains $h(x)$. The invariant of blocked probing is that each key is stored as close as possible to $h(x)$ in the sense that $i_x \in I(x, j)$ if there is sufficient space, i.e., if the number of keys with hash values in $I(x, j)$ is at most $|I(x, j)| = 2^j$. Below we describe the operations of blocked probing.

When *inserting* a key $x$, the invariant is maintained by searching, for $j = 0, 1, 2, \dots$, for a location $i \in I(x, j)$ where $x$ could be placed. For each $j$, we first check if there is an empty location in $I(x, j)$ and put $x$ there if there is one. Otherwise, we look for a location $i_{x'} \in I(x, j)$ that contains a key $x'$ with $d(x', i_{x'}) > j$ (implying that $h(x') \notin I(x, j)$). If there is such an $x'$, we swap $x$ and $x'$, and continue the insertion process with $x'$. If both attempts fail, we move on to the next $j$.

A *search* for $x$ proceeds by inspecting, for $j = 0, 1, 2, \dots$, the locations of $I(x, j)$ until either $x$ is found, or we do not find $x$ but find instead an empty location or a key $x'$ with $d(x', i_{x'}) > j$. In the latter cases, the invariant tells us that $x$ is not present in the hash table.

*Deletion* of a key $x \in I(x, j) \backslash I(x, j-1)$ needs to check if there is a key stored in $I(x, j+1) \backslash I(x, j)$ that could be stored in $I(x, j)$ — if this is the case it is copied to the empty location, and the old copy is deleted recursively.

## 4.2 Cache-oblivious analysis of blocked probing

We assume the block size $b$ is a power of two, and the $i$-th block $\mathcal{B}_i$ starts at position $ib$ and ends at position $ib+b-1$. Then for any key $x$, the aligned block $I(x, \log b)$ is the block that contains $h(x)$. Let $S$ denote the set of keys involved in a given operation (insertion, deletion, successful or unsuccessful search), including the key $x$ specified by the query or update ($x$ may or may not be in the hash table). To bound the expected I/O cost for an operation, define event $\mathcal{E}_s(x, j)$ as the aligned block $I(x, j)$ being *saturated*, that is, the number of keys in $S$ with hash value in the aligned block $I(x, j)$ is $2^j$ or more. Let $p(x, j)$ denote the probability that $\mathcal{E}_s(x, j)$ happens. The following lemma relates $p(x, j)$ with $C_{bp}$, the expected I/O cost for an operation of blocked probing.

**Lemma 1** *Suppose function $h$ is drawn from a pairwise independent hash family $\mathcal{H}$, then*

$$C_{bp} \le 1 + \sum_{j=1+\log b}^{\log r} \frac{2^{j+2}}{b} p(x, j).$$

*Proof*: We first note that the cost of a search for key $x$ is bounded by that of an insertion of $x$, so we only need to consider insertions and deletions. Let $\mathcal{E}_f(x, j)$ denote the event that the aligned block $I(x, j)$ is full, that is, the number of keys stored in $I(x, j)$ is $2^j$. Let $q(x, j)$ denote the probability that $\mathcal{E}_f(x, j)$ happens. Observe that an insertion or a deletion would visit a location outside $I(x, j)$ only if all positions of $I(x, j)$ are occupied, so the probability that the operation takes at least $2^j/b$ I/Os is $q(x, j)$, for $j \ge \log b$. To compute the expected number of blocks involved in an operation, in addition to the first I/O that retrieves $I(x, \log b)$, we first bound the sum of the probabilities that the operation takes at least $i$ I/Os, for $i$ from $2^j/b$ to $2^{j+1}/b - 1$ for a fixed integer $j \ge \log b$:

$$\sum_{i=2^j/b}^{i=2^{j+1}/b-1} \Pr[\text{Operation takes at least } i \text{ I/Os}] \le \sum_{i=2^j/b}^{i=2^{j+1}/b-1} q(x, j) = \frac{2^j}{b} q(x, j).$$

Summing over all possible values of $j > \log b$ and we have

$$\begin{aligned} C_{bp} &= 1 + \sum_{j=\log b}^{\infty} \sum_{i=2^j/b}^{i=2^{j+1}/b-1} \Pr[\text{Operation takes at least } i \text{ I/Os}] \\ &\le 1 + \sum_{j=1+\log b}^{\infty} \frac{2^j}{b} q(x, j). \end{aligned} \tag{2}$$

Next we will relate $q(x, j)$, the probability that $I(x, j)$ is full, with $p(x, j)$, the probability that $I(x, j)$ is saturated. Divide the hash table $r$ into $\log(r/2^b) + 1$ aligned blocks:

$$\mathcal{I} = \{I(x, j), I(x, j+1) \setminus I(x, j), I(x, j+2) \setminus I(x, j+1), \ldots, I(x, r) \setminus I(x, r/2)\}.$$

The claim is that if $I(x, j)$ is full, then at least one of the aligned blocks in $\mathcal{I}$ is saturated. For a proof, assume that no aligned block in $\mathcal{I}$ is saturated. We inductively prove that each aligned block in $\mathcal{I}$ only stores keys with hash values inside it, which immediately implies that $I(x, j)$ is non-full, and thus leads to a contradiction. For the first insertion the statement is true. Now suppose the statement is true after the $k$-th insertion. When the $(k+1)$-th insertion $y_{k+1}$ comes, let $I(x, l+1) \setminus I(x, l)$ denote the aligned block in $\mathcal{I}$

that contains $h(y_{k+1})$. By the inductive hypothesis, $I(x, l+1) \setminus I(x, l)$ only contains the keys with hash values in it, and since $I(x, l+1) \setminus I(x, l)$ is not saturated we know that $I(x, l+1) \setminus I(x, l)$ is non-full. Therefore the key $y_{k+1}$ is stored in an empty position of $I(x, l+1) \setminus I(x, l)$, and the induction follows.

Observe that since the hash function $h$ is drawn from a pairwise independent family and the fact that $I(x, l+1) \setminus I(x, l)$ and $I(x, l)$ are of the same size, the probability that the $I(x, l+1) \setminus I(x, l)$ is saturated is the same as the probability that $I(x, l)$ is saturated, that is, $p(x, l)$. By a union bound we have the following inequality:

$$q(x, j) \leq p(x, j) + \sum_{l=j}^{\log r} p(x, l). \tag{3}$$

Combining (2) and (3) we have

$$\begin{aligned} C_{bp} &\leq 1 + \sum_{j=1+\log b}^{\log r} \frac{2^j}{b} q(x, j) \\ &\leq 1 + \sum_{j=1+\log b}^{\log r} \frac{2^j}{b} \left( p(x, j) + \sum_{l=j}^{\log r} p(x, l) \right) \\ &= 1 + \sum_{j=1+\log b}^{\log r} \frac{1}{b} \left( 2^j + \sum_{l=1+\log r}^{j} 2^l \right) p(x, j) \\ &\leq 1 + \sum_{j=1+\log b}^{\log r} \frac{2^{j+2}}{b} p(x, j). \end{aligned}$$

□

For a truly random hash function, $p(x, j)$ can bounded using the Chernoff bound: The probability that a key is hashed to $I(x, j)$ is $2^j/r$, so the expected number of keys hashed to $I(x, j)$ is $n2^j/r = \alpha 2^j$. Recall that $p(x, j)$ is the probability that the number of keys hashed to $I(x, j)$ is $2^j$ or more, by the Chernoff bound, $p(x, j) \leq 2^{-(1-\alpha)^2 2^{j-1}}$. Following Lemma 1, we have

$$\begin{aligned} C_{bp} &\leq 1 + \sum_{j=1+\log b}^{\log r} 2^{j+2} p(x, j) \\ &= 1 + \sum_{j=1+\log b}^{\log r} (2^{j+2}/b) 2^{-(1-\alpha)^2 2^{j-1}} \\ &\leq 1 + 2^{-\Omega((1-\alpha)^2 b)}. \end{aligned}$$

That $h$ is a truly random hash function is an unrealistic assumption. To analyze blocked probing with limited independence, we need the following variant of the Chernoff bound by Schmidt et al. [23]:

**Lemma 2 ([23])** *Let $X_1, \ldots, X_n$ be a sequence of $k$-wise independent random variables, that satisfy $|X_i - E[X_i]| \leq 1$. Let $X = \sum_{i=1}^n X_i$ with $\mathrm{E}[X] = \mu$, and let $\delta^2[X]$ denote the variance of $X$, so that $\delta^2[X] = \sum_{i=1}^n \delta^2[X_i]$ (this equation holds provided $k \geq 2$). Then for any even $k$ and $C \geq \max\{k, \delta^2[X]\}$,*

$$\Pr[|X - \mu| \geq T] \leq \left( \frac{kC}{e^{2/3} T^2} \right)^{k/2}.$$

Lemma 1 and 2 together will lead to the following result:

**Theorem 2** *Consider a blocked probing hash table in the cache-oblivious model where the block size $b$ is power of $2$ and every block starts at a memory address divisible by $b$. Suppose the hash table has a fixed size $r$ and the hash function $h$ is chosen uniformly at random from a $k$-wise independent hash family for odd $k \geq 5$. For any sequence of operations (insertions, deletions, and lookups), let $\alpha$ denote the load factor of the hash table during a particular operation. Then the expected number of I/Os for that operation is*

$$C_{bp} = 1 + O\left(\left(\frac{k-1}{e^{2/3}(1-\alpha)^2 b}\right)^{(k-1)/2}\right).$$

*Proof*: Consider an operation on key $x$. We need to bound $p(x,j)$, the probability that the aligned block $I(x,j)$ is saturated, for $j \geq \log b$. Let $X_i$ denote the random variable indicating that the $i$-th key has hash value in $I(x,j)$. Note that $X_1, \ldots, X_n$ are $(k-1)$-wise independent, and for each $X_i$ we have $\mathrm{E}[X_i] = 2^j/r$ and $\delta^2[X_i] = 2^j/r(1-2^j/r) \leq 2^j/r$. It follows that $\mathrm{E}[X] = \sum_{i=1}^n \mathrm{E}[X_i] = 2^j n/r = \alpha 2^j$ and $\delta^2[X] = \sum_{i=1}^n \delta^2[X_i] \leq 2^j n/r = \alpha 2^j$. Setting $\mu = \alpha 2^j$, $T = (1-\alpha)2^j$, $C = 2^j \geq \max\{k, \delta^2[X]\}$ in Lemma 2, we derive a bound on $p(x,j)$:

$$p(x,j) = \Pr[X - \alpha 2^j \geq (1-\alpha)2^j] \leq \left(\frac{k-1}{e^{2/3}(1-\alpha)^2 2^j}\right)^{(k-1)/2}. \tag{4}$$

Plugging (4) into Lemma 1:

$$\begin{aligned} C_{bp} &\leq 1 + \sum_{j=1+\log b}^{\log r} (2^{j+2}/b)p(x,j) \\ &\leq 1 + \sum_{j=1+\log b}^{\log r} \frac{2^{j+2}}{b} \cdot \left(\frac{k-1}{e^{2/3}(1-\alpha)^2 2^j}\right)^{(k-1)/2} \\ &\leq 1 + O\left(\left(\frac{k-1}{e^{2/3}(1-\alpha)^2 b}\right)^{(k-1)/2}\right). \end{aligned}$$

The last inequality uses that fact that the terms in the sum are geometrically decreasing when $k \geq 5$, and hence the sum is dominated by the first term. $\square$

**Remark:** Since a $k$-wise independent hash function is also $k'$-wise independent for all $k' \geq k$, the bound in Theorem 2 is actually $1 + O\left(\min_{5 \leq k' \leq k} \left(\frac{k'-1}{e^{2/3}(1-\alpha)^2 b}\right)^{(k'-1)/2}\right)$.

Theorem 2 immediately leads to the following corollaries.

**5-wise independence** The minimum independence allowed in Theorem 2 is 5. In this case

$$C_{bp} = 1 + O\left(\frac{1}{b^2}\right).$$

Note that the dependence on the block size $b$ is asymptotically better than $1 + \Theta(1/b)$.

**$\Omega(b)$-wise independence** To achieve the same bound as that of the truly random hash function, it suffices to have $k \geq k' = (1-\alpha)^2 b + 1$. By Theorem 2, it follows that

$$\begin{aligned} C_{bp} &= 1 + O\left(\left(\frac{k'-1}{e^{2/3}(1-\alpha)^2 b}\right)^{(k'-1)/2}\right) \\ &= 1 + O\left(\left(e^{2/3}\right)^{-(1-\alpha)^2 b/2}\right) \\ &\leq 1 + 2^{-\Omega((1-\alpha)^2 b)}. \end{aligned}$$

## 4.3 Cache-oblivious dynamic hash tables

The standard doubling/halving strategy can be used to maintain the load factor $\alpha$ in the range $1/2 - \varepsilon/2 \leq \alpha \leq 1 - \varepsilon$ as we insert and delete keys in the hash table where $\varepsilon > 0$ is any small constant. In such a range the expected I/O cost per operation is $1 + 1/2^{\Omega(b)}$ I/Os using the blocked probing scheme described above. In particular, we always use a hash table of size $r$ that is a power of 2. Let $g : [u] \to [u]$ be a "mother" hash function. When the table's size is $r$, we take the $\log r$ least significant bits of $g(x)$ as $h(x)$. When $\alpha = n/r$ goes beyond the range $[1/2 - \varepsilon/2, 1 - \varepsilon]$ we double or halve $r$ accordingly. This can be done in a simple scan of the hash table in amortized $O(1/b)$ I/Os per key, by simply inserting keys in the order they occur in the table. The analysis uses the fact that the keys to be inserted in a block in the resized hash table are (w.h.p.) in at most two blocks in the original hash table. We omit the rather standard analysis.

However, the above solution has a poor space utilization. A number of methods have been proposed that maintain a higher load factor, and also allow the rehashing to be done incrementally; see [15] for an overview. To our best knowledge these methods are all cache-aware — however, we now describe how they can be made cache-oblivious while maintaining the load factor of $\alpha = 1 - \Theta(\varepsilon)$. Suppose initially $r$ is a power of 2 and $n > (1 - 2\varepsilon)r$. Adjust $\varepsilon$ so that $\varepsilon r$ is also a power of 2; this will not change $\varepsilon$ by more than a factor of 2. The idea is to split the hash table into $1/\varepsilon$ parts using hashing (say, by looking at the first $\log(1/\varepsilon)$ bits of the mother hash function), where each part is handled by a cache-oblivious hash table of size $\varepsilon r$ which stores at most $(1-\varepsilon)\varepsilon r$ keys. As $n$ changes, the number of parts also changes to maintain the overall load factor at $\alpha = 1 - \Theta(\varepsilon)$. Now this situation is analogous to a standard cache-aware hash table with "block size" being equal to $(1-\varepsilon)\varepsilon r$, and parts corresponding to blocks. So we may use any cache-aware method that resizes a standard hash table, such as *linear hashing* [17]. These resizing techniques will split or merge parts as needed, and cost is $O(1/b)$ I/Os per insertion/deletion amortized. When $r$ doubles or halves, we rebuild the entire hash table using a new part size $\varepsilon r$. The cache-aware resizing techniques ensures that only $1 + 1/2^{\Omega(b')}$ parts are accessed upon a query in expectation, where $b'$ is the part size $b' = (1-\varepsilon)\varepsilon r$. Within each part, our cache-oblivious scheme accesses $1 + 1/2^{\Omega(b)}$ blocks. So as long as $r \gg b$, the overall query cost is still $1 + 1/2^{\Omega(b)}$ I/Os, as desired.

In summary, we can dynamically update our cache-oblivious hash table while maintaining a high load factor. The additional resizing cost is only $O(1/b)$ I/Os amortized.

**Theorem 3** *In the cache-oblivious model where the block size $b$ is a power of 2 and every block starts at a memory address divisible by $b$, there is a dynamic hash table that supports queries in expected average $t_q = 1 + 1/2^{\Omega(b)}$ I/Os, and insertions and deletions of keys in expected amortized $1 + O(1/b)$ I/Os. The load factor can be maintained at $\alpha \geq 1 - \varepsilon$ for any constant $\varepsilon > 0$.*

**Remark** If a $k$-wise independent hash family is used, the bound on $t_q$ in the above theorem will be replaced by the bound in Theorem 2.

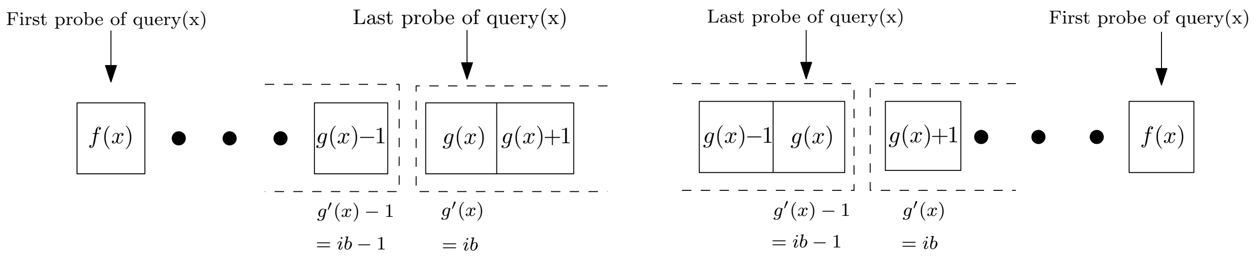

Figure 1: When two I/Os are needed.

## 5 Lower Bounds

In this section, we show that the two conditions that the analysis of blocked probing depends upon are both necessary to achieve a $1+1/2^{\Omega(b)}$ search cost. Specifically, we prove that when either condition is removed, the best obtainable bound for the expected average cost of a successful search is $1+O(\alpha/b)$ I/Os. The lower bound proofs allow $\alpha$ to be asymptotically small, so it means that we cannot hope to do a lot better even with super-linear space.

### 5.1 The model

Before we present the exact lower bound statements let us first be more precise about our model. Let $U=[u]$ be the universe. The hard input we consider here is a random input in which each key is drawn from $U$ uniformly and independently. Let $I_{\mathbf{u}}$ be such a random input, and $\mathcal{I}$ be the set of all inputs. We will bound from below the expected average cost of a successful search on $I_{\mathbf{u}}$ where the average is taken over all keys in $I_{\mathbf{u}}$. We will only consider deterministic hash tables; the lower bounds also hold for randomized hash tables by invoking *Yao's minimax principle* [19] because we are using a random input. The hash table can employ any hash functions to distribute the input. We assume $u>n^3$, then with probability $1-O(1/n)$ all keys in $I_{\mathbf{u}}$ are distinct by the birthday paradox.

We assume that all the $n$ keys are stored in a table of size $r$ on external memory[3], possibly with duplication. We model the search algorithm by two functions $f,g:[u]\to[r]$. For any $x\in[u]$, $f(x)$ is the position where the algorithm makes its first probe, while $g(x)$ is the position of the last probe, where key $x$ (or one of its copies) must be located. Note that the internal memory must be able to hold the description of $f$, thus any deterministic hash table can employ a family $\mathcal{F}$ of at most $2^{m\log u}$ such functions. Although the particular $f$ used by the hash table of course can depend on the input $I_{\mathbf{u}}$, the family $\mathcal{F}$ has to be fixed in advance. We do not have any restrictions on $g$, as it is possible for the search algorithm to evaluate $g$ after accessing external memory, except that all $g(x)$'s are distinct for the $n$ keys.

The table is partitioned into blocks of size $b$. For any $x$ such that $f(x)\neq g(x)$, define $g'(x)$ to be $g(x)$ if $f(x)<g(x)$ and $g(x)+1$ if $f(x)>g(x)$. Then if $g'(x)$ is the first position of a block, at least two blocks must have been accessed, though the reverse is not necessarily true; please refer to Figure 1. For lower bound purposes we will assume optimistically that the search for $x$ needs two I/Os if $g'(x)$ is the first position of a block, and one I/O otherwise. Note that after this abstraction, the search cost is completely characterized by the functions $f,g$ and the blocking.

[3]Here we do not allow keys to be stored in internal memory: since the memory holds at most $m$ keys, it does not affect the average search cost as long as $n$ is sufficiently larger than $m$.

We will consider the following two blocking models. In the *boundary-oblivious* model, the hash table knows the block size $b$ but not their boundaries. More precisely, how the keys are stored in the table is allowed to depend on $b$, but the layout should work for any shifting $s$, namely when each block spans the positions from $ib - s$ to $(i+1)b - s - 1$ for $s = 0, 1, \dots, b-1$. In the *block-size-oblivious* model, the blocks always start at positions that are multiples of $b$ but the layout is required to work for all $b = 1, \dots, r$. Below we will show that in either model, the best possible expected average cost of a successful search is $1 + O(\alpha/b)$ I/Os.

## 5.2 Good inputs and bad inputs

For any $I \in \mathcal{I}, f \in \mathcal{F}$, define $\eta_f(I) = \sum_{i \in [r]}(|\{x \in I \mid f(x) = i\}| - 1)$. Intuitively, $\eta_f(I)$ is the number of the overflowed keys; since each position $i$ can only hold one key, at least $\eta_f(I)$ keys in $I$ need a second probe when the hash table uses $f$ to decide its first probe. We say an input $I \in \mathcal{I}$ is *bad* with respect to $f$ if $\eta_f(I) \geq \frac{\alpha}{4}n$, otherwise it is *good*. Let $\mathcal{I}_f$ be the set of all bad inputs with respect to $f$, and $\mathcal{I}_\mathcal{F} = \bigcap_{f \in \mathcal{F}} \mathcal{I}_f$ which is the set of inputs that are bad with respect to all $f \in \mathcal{F}$. In our lower bounds we will actually focus only on the bad inputs $\mathcal{I}_\mathcal{F}$. The following technical lemma ensures that almost all inputs are in $\mathcal{I}_\mathcal{F}$.

**Lemma 3** *For $n > cm \log u/\alpha^2$ where $c$ is some sufficiently large constant and $\alpha = \omega(n^{-1/2})$, $I_\mathbf{u}$ is a bad input with respect to all $f \in \mathcal{F}$ with probability $1 - o(1)$ as $n \to \infty$.*

The general idea of the proof is the following: We first show that for a particular $f$ and a random $I_\mathbf{u}$, the probability that $I_\mathbf{u}$ is good with respect to $f$ is $e^{-\Omega(\alpha^2 n)}$. Thus by a union bound, $I_\mathbf{u}$ is good for at least one $f \in \mathcal{F}$ with probability at most $e^{-\Omega(\alpha^2 n)} \cdot 2^{m \log u}$. So as long as $n$ is large enough, $I_\mathbf{u}$ will be bad with respect to all $f \in \mathcal{F}$ with high probability.

We need the following bin-ball game, which models the way how $f$ works on a uniformly random input:

**A bin-ball game** In a $(n, r, \vec{\beta})$ *bin-ball game*, we throw $n$ balls into $r$ bins independently at random. The probability that a ball goes to the $j$-th bin is $\beta_j$, where $\vec{\beta} = (\beta_0, \dots, \beta_{r-1})$ is a prefixed distribution. Let $Z$ denote the number of empty bins after $n$ balls are thrown in.

**Lemma 4** *In an $(n, r, \vec{\beta})$ bin-ball game,* $\Pr[Z \leq r - n + \frac{\alpha}{4}n] \leq e^{-\Omega(\alpha^2 n)}$, *where $\alpha = n/r$.*

*Proof*: Note that if $\vec{\beta}$ is the uniform distribution, the problem is known as the *occupancy problem* and the lemma can be proved using properties of martingales [19]. The same proof actually also holds for a nonuniform $\vec{\beta}$, so we just sketch it here:

Let $Z_0$ be the expectation of $Z$ before any ball is thrown in, and let the random variable $Z_i$ be the expectation of $Z$ after the $i$-th ball is thrown in, for $i = 1, \dots, n$. Note that $Z_i, i \geq 1$ is a random variable, where randomness comes from the first $i$ balls. In particular we have $Z_0 = \mathrm{E}[Z]$ and $Z_n = Z$. It can be verified that the sequence $Z_0, Z_1, \dots, Z_n$ is a martingale, and that $|Z_{i+1} - Z_i| \leq 1$ for all $0 \leq i < n$. Therefore by Azuma's inequality, we get

$$\Pr[Z \leq \mathrm{E}[Z] - \lambda n^{1/2}] \leq 2e^{-\lambda^2/2}.$$

Note that

$$\mathrm{E}[Z] \;=\; \sum_{i=0}^{r-1}(1-\beta_i)^n \geq r\left(\frac{r - \sum_{i=0}^{r-1}\beta_i}{r}\right)^n = r\left(1 - \frac{1}{r}\right)^n$$

$$\geq \quad r - n + \frac{\alpha}{2}n - \frac{\alpha}{2} - \frac{(n-1)(n-2)}{6n}\alpha^2.$$

Setting $\lambda = (\frac{\alpha}{4}n - \frac{\alpha}{2} - \frac{(n-1)(n-2)}{6n}\alpha^2)n^{-1/2} = \Omega(\alpha n^{1/2})$, we have $\mathrm{E}[Z] - \lambda n^{1/2} \geq r - n + \frac{\alpha}{4}n$, hence the lemma. $\Box$

Now we are ready to prove Lemma 3.

*Proof*:(of Lemma 3) Consider a particular $f : [u] \to [r]$ and a random input $I_{\mathbf{u}}$. The probability that a randomly chosen key $x$ from $[u]$ has $f(x) = i$ is exactly $|f^{-1}(i)|/u$. This is exactly an $(n, r, \vec{\beta})$ bin-ball game where $\beta_i = |f^{-1}(i)|/u$. Let $Z$ be the number of empty bins at the end of such a bin-ball game. Note that we have $\eta_f(I) = n - (r - Z)$, which, by Lemma 3, does not exceed $\frac{\alpha}{4}n$ with probability at most $e^{-\Omega(\alpha^2 n)}$. Since there are $2^{m \log u}$ different $f$'s in $\mathcal{F}$, by a union bound, the probability that $I_{\mathbf{u}}$ is good for at least one $f \in \mathcal{F}$ is at most $e^{-\Omega(\alpha^2 n)} \cdot 2^{m \log u}$. Thus if $n > cm \log u/\alpha^2$ for some sufficiently large $c$, this probability is $e^{-\Omega(\alpha^2 n)} = o(1)$. $\Box$

## 5.3 Lower bound for the boundary-oblivious model

Now we prove the lower bound for the boundary-oblivious model, where the layout is required to work for any shifting $s$.

**Theorem 4** *For any fixed block size $b$, consider any hash table that stores $n$ uniformly random keys. There exists some shifting $s$ for which the hash table has an expected average successful search cost at least $1 + \frac{\alpha}{5b}$, for $n$ sufficiently large and $\alpha = \omega(n^{-1/2})$.*

*Proof*: Consider any input $I \in \mathcal{I}$. Suppose that the hash table uses $f_I \in \mathcal{F}$ and $g_I$ on input $I$. Define $\gamma(s, I)$ to be the number of keys in $I$ that need at least two I/Os to search when the shifting is $s$, i.e., those keys $x$ with $f_I(x) \neq g_I(x)$ and $g'_I(x) = ib - s$ for some integer $i$. Note that the average search cost on $I$ is at least $1 + \gamma(s, I)/n$, and the expected average search cost on a random $I_{\mathbf{u}}$ is at least $1 + \mathrm{E}_{\mathbf{u}}[\gamma(s, I_{\mathbf{u}})]/n$, which we will show to be greater than $1 + \frac{\alpha}{5b}$.

Consider any $I \in \mathcal{I}_{\mathcal{F}}$. Since $I$ is bad for all $f \in \mathcal{F}$, it is also bad for $f_I$. Thus there are at least $\frac{\alpha}{4}n$ keys $x$ in $I$ with $f_I(x) \neq g_I(x)$. For these keys, $g'_I(x)$ is defined and there is exactly one $s$ such that $g'_I(x) = ib - s$ for some integer $i$. To show that there is a shift $s$ with large search cost on average, we sum up $\gamma(s, I)$ for $s$ from 0 to $b - 1$ and get $\sum_{s=0}^{b-1} \gamma(s, I) \geq \frac{\alpha}{4}n$. By Lemma 3, $I_{\mathbf{u}}$ belongs to $\mathcal{I}_{\mathcal{F}}$ with probability $1 - o(1)$, so

$$\sum_{s=0}^{b-1} \mathrm{E}_{\mathbf{u}}[\gamma(s, I_{\mathbf{u}})] = \mathrm{E}_{\mathbf{u}}\left[\sum_{s=0}^{b-1} \gamma(s, I_{\mathbf{u}})\right] \geq (1 - o(1))\frac{\alpha}{4}n \geq \frac{\alpha}{5}n.$$

By the pigeonhole principle, we must have one $s$ such that $\mathrm{E}_{\mathbf{u}}[\gamma(s, I_{\mathbf{u}})] \geq \frac{\alpha n}{5b}$, and the lemma is proved. $\Box$

## 5.4 Lower bound for the block-size-oblivious model

Next we give the lower bound under the block-size-oblivious model, in which the block boundaries are always multiples of $b$, but the layout of the hash table is required to work with any $b$. Since it is not possible to prove a lower bound of the form $1+\Omega(\alpha/b)$ for all $b$ (that would be a lower bound in the cache-aware model), instead we show that $1 + o(\alpha/b)$ is not achievable, i.e., the following is false: "$\forall\epsilon\exists n_0\exists b_0\forall n > n_0\forall b > b_0$, the cost is at most $1 + \epsilon\alpha/b$." In particular, we show that this statement is false for $\epsilon = \frac{1}{17}$.

**Theorem 5** *Consider any hash table that stores $n$ uniformly random keys. For any $b_0$, there exists a block size $b \geq b_0$ on which the expected average success search cost on $n$ keys is at least $1 + \frac{\alpha}{17b}$, for any $n$ sufficiently large and $\alpha = \omega((\log \log n)^{-1/2})$.*

We follow the same framework as in the proof of Theorem 4. Let $\rho(b, I)$ be the number of keys $x$ in $I$ with $f_I(x) \neq g_I(x)$ and $b|g'_I(x)$; these keys need two I/Os to search when the block size is $b$ in the block-size-oblivious model. On a random $I_u$, the expected average search cost is $1 + \mathrm{E}_{\mathbf{u}}[\rho(b, I_{\mathbf{u}})]/n$. From here suppose we were to continue to follow the proof of Theorem 4 and consider the summation of $\mathrm{E}_{\mathbf{u}}[\rho(b, I_{\mathbf{u}})]$ over all $b \in \{b_0, b_0 + 1, \dots, r\}$. Each $x$ contributes 1 to the summation when $b = g'_I(x)$, so we still have $\sum_{b=b_0}^{r} \mathrm{E}_{\mathbf{u}}[\rho(b, I_{\mathbf{u}})] = \Omega(\alpha n)$. This, unfortunately, only guarantees the existence of a $b$ such that $\mathrm{E}_{\mathbf{u}}[\rho(b, I_{\mathbf{u}})]$ is at least $\Omega(\frac{\alpha n}{r})$ or $\Omega(\frac{\alpha n}{b \log r})$, where the latter uses the fact that $\sum_{b=b_0}^{r} 1/b = \Theta(\log r)$. Neither is strong enough to give us the desired lower bound. Below we show how we prove Theorem 5 by restricting $b$ to the primes and a much more careful analysis.

**Lemma 5** *Let $P_k$ be the set of all primes that are smaller than $k$, and let $P = P_r - P_{b_0}$ be the set of all primes that are in the range $[b_0, r)$. For $\alpha = \omega((\log \log n)^{-1/2})$, we have*

$$\mathrm{E}_{\mathbf{u}}\left[\sum_{b \in P} \rho(b, I_{\mathbf{u}})\right] = \sum_{b \in P} \mathrm{E}_{\mathbf{u}}[\rho(b, I_{\mathbf{u}})] > (1 - o(1))\frac{\alpha}{16} n \log \log r,$$

*as $n \to \infty$.*

Note that Lemma 5 implies that there must be a $b \in P$ such that $\mathrm{E}[\rho(b, I_{\mathbf{u}})] \geq \frac{\alpha}{17b} n$, proving Theorem 5, since otherwise we would have

$$\sum_{b \in P} \mathrm{E}[\rho(b, I_{\mathbf{u}})] \leq \frac{\alpha}{17} n \sum_{b \in P} \frac{1}{b} \leq \frac{\alpha}{17} n (\log \log r + O(1)).$$

Here we use the following approximation for the prime harmonic series [24]:

$$\sum_{b \in P_r} \frac{1}{b} = \log \log r + O(1).$$

Thus $\sum_{b \in P} \mathrm{E}[\rho(b, I_{\mathbf{u}})] \leq \frac{\alpha}{17} n \left(\log \log r + O(1)\right)$, contradicting Lemma 5.

**Proof of Lemma 5** In the rest of this subsection we prove Lemma 5. We need the following fact from number theory. Let $\mu(s)$ denote the number of distinct prime factors of $s$.

**Lemma 6 ([24])** *Let $\xi(r) \to \infty$. Then*

$$\left|\left\{ l \leq r : |\mu(l) - \log \log r| > \xi(r)\sqrt{\log \log r} \right\}\right| = O\left(\frac{r}{\xi^2(r)}\right).$$

*Proof*:(of Lemma 5) By Lemma 3 we know that $I_{\mathbf{u}}$ belongs to $\mathcal{I}_{\mathcal{F}}$ with probability $1 - o(1)$, so it suffices to prove that for any $I \in \mathcal{I}_{\mathcal{F}}$,

$$\sum_{b \in P} \rho(b, I) > (1 - o(1))\frac{\alpha}{16} n \log \log r.$$

Consider any $I \in \mathcal{I}_{\mathcal{F}}$. Let $G$ be the set of distinct $g'_I(x)$'s for the keys $x \in I$. Let $\mu_P(s)$ be the number of distinct prime factors of $s$ that are in $P$. By definition $\mu_{P_{b_0}}(s)$ is the number of distinct prime factors of $s$ that are in $P_{b_0}$, and it follows that $\mu(s) = \mu_{P_{b_0}}(s) + \mu_P(s)$. Note that $\rho(b, I)$ is at least the number of multiples of $b$ in $G$, so we have

$$\sum_{b \in P} \rho(b, I) \geq \sum_{l \in G} \mu_P(l) = \sum_{l \in G} \mu(l) - \sum_{l \in G} \mu_{P_{b_0}}(l). \tag{5}$$

Next we show that $\sum_{l \in G} \mu(l)$ is large. Firstly, observe that

$$|G| > \frac{\alpha}{8} n. \tag{6}$$

This is because $I$ is bad for $f_I$, so at least $\frac{\alpha}{4}n$ keys in $I$ have $f_I(x) \neq g_I(x)$ and thus their $g'_I(x)$'s are defined. The $g_I(x)$'s for these keys must be distinct, and each $g'_I(x)$ is either $g_I(x)$ or $g_I(x) + 1$, so there are at least $\frac{\alpha}{8}n$ distinct $g'_I(x)$'s for the keys in $I$.

Secondly, by choosing $\xi(r) = \frac{(\log\log r)^{1/4}}{\sqrt{\alpha}}$ in Lemma 6 we get:

$$\begin{aligned} &\left| \left\{ l \leq r : \mu(l) \leq \left(1 - \frac{1}{\sqrt{\alpha}(\log\log r)^{1/4}}\right) \log\log r \right\} \right| \\ &= O\left( \frac{\alpha r}{\sqrt{\log\log r}} \right). \end{aligned}$$

Since we require $\alpha = \omega(\frac{1}{\sqrt{\log\log n}})$ which implies $\frac{\alpha r}{\sqrt{\log\log r}} = \frac{\alpha n}{\alpha\sqrt{\log\log r}} = o(\frac{\alpha}{8}n)$ and $\frac{1}{\sqrt{\alpha}(\log\log r)^{1/4}} = o(1)$, it holds that for at least $|G| - o(1)\frac{\alpha}{8}n$ distinct $l \in G$,

$$\mu(l) > (1 - o(1)) \log\log r. \tag{7}$$

By inequalities (6) and (7), we have

$$\sum_{l \in G} \mu(l) > (1 - o(1)) \frac{\alpha}{8} n \log\log r. \tag{8}$$

It remains to upper bound $\sum_{l \in G} \mu_{P_{b_0}}(l)$. Note that for any $b \in P_{b_0}$, the number of integers in $[r]$ that are divisible by $b$ is at most $r/b$, so each $b$ will be counted at most $r/b$ times in $\sum_{l \in G} \mu_{P_{b_0}}(l)$. Hence,

$$\sum_{l \in G} \mu_{P_{b_0}}(l) \leq \sum_{b \in P_{b_0}} r/b = r\left(\log\log b_0 + O(1)\right).$$

Therefore, as long as $\alpha \geq \sqrt{\frac{32 \log\log b_0}{\log\log n}} > \sqrt{\frac{16 \log\log b_0}{\log\log n/\alpha}}$, we have

$$\log\log b_0 < \frac{\alpha^2}{16} \log\log \frac{n}{\alpha},$$

so

$$\begin{aligned} \sum_{l \in G} \mu_{P_{b_0}}(l) &< \frac{\alpha}{16} n \log\log r + O(r) \\ &= (1 + o(1)) \frac{\alpha}{16} n \log\log r. \end{aligned} \tag{9}$$

Finally, combining (5), (9), and (8) completes the proof. □

### 5.5 Lower bounds on updates

Our lower bounds in this paper are concerned with the query cost only. How about updates? The blocked probing algorithm in Section 4 has an amortized update cost of $1 + O(1/b)$ I/Os, but can we improve it to $o(1)$ I/Os, possibly by buffering the updates in internal memory and write them to external memory in batches? A recent result by Wei et al. [27] has eliminated this possibility by proving a $1 - 1/2^{\Omega(b)}$ lower bound (in the cache-aware model) on the amortized update cost if the successful query cost is to be $t_q = 1 + 1/2^{\Omega(b)}$. Even more recently, Verbin and Zhang proved [25] that if $t_q$ is $o(\log_{b \log n} n)$ for both successful and unsuccessful queries, then the amortized update cost has to be at least $0.99$ I/O. These results show that for external hashing, buffering is essentially useless and modifying the hash table on disk directly is the only way to perform updates.

## 6 Open Problems

An interesting open question is, although we have proved a matching lower bound in the cache-oblivious model, we do not yet know if $t_q = 1+1/2^{\Omega(b)}$ is optimal in the cache-aware model (or in the cache-oblivious model with the two more conditions). It is known that we can achieve $t_q = 1$ (namely, *perfect hashing*) with an internal memory of size $m = \Theta(n/b)$ [11, 16, 18]. On the other hand, external linear probing and blocked probing achieve $t_q = 1 + 1/2^{\Omega(b)}$ with only $m = \Theta(b)$. There seems to be a tradeoff between $m$ and $t_q$ but this tradeoff is yet to be understood.

## References

[1] P. Afshani, C. Hamilton, and N. Zeh. Cache-Oblivious Range Reporting with Optimal Queries Requires Superlinear Space. *Discrete and Computational Geometry*, 45(4):824–850, 2011.

[2] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[3] M. A. Bender, G. S. Brodal, R. Fagerberg, D. Ge, S. He, H. Hu, J. Iacono, and A. López-Ortiz. The Cost of Cache-Oblivious Searching. *Algorithmica*, 61(2):463–505, 2010.

[4] M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-oblivious B-trees. *SIAM Journal on Computing*, 35(2):341–358, 2005.

[5] G. S. Brodal and R. Fagerberg. On the limits of cache-obliviousness. In *Proc. ACM Symposium on Theory of Computing*, pages 307–315, 2003.

[6] J. Carter and M. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[7] E. Demaine. Cache-oblivious algorithms and data structures. In *EEF Summer School on Massive Datasets*. Springer Verlag, 2002.

[8] R. Fagin, J. Nievergelt, N. Pippenger, and H. Strong. Extendible hashing—a fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

[9] M. L. Fredman, J. Komlos, and E. Szemeredi. Storing a sparse table with $o(1)$ worst -case access time. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 165–170, 1982.

[10] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 285–298, 1999.

[11] G. H. Gonnet and P.-Å. Larson. External hashing with limited internal storage. *Journal of the ACM*, 35(1):161–184, 1988.

[12] B. He and Q. Luo. Cache-oblivious databases: Limitations and opportunities. *ACM Transactions on Database Systems*, 33(2), Article 8, 2008.

[13] M. S. Jensen and R. Pagh. Optimality in external memory hashing. *Algorithmica*, 52(3):403–411, 2008.

[14] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 1973.

[15] P.-A. Larson. Dynamic hash tables. *Communications of the ACM*, 31(4):446–457, 1988.

[16] P.-A. Larson. Linear hashing with separators—a dynamic hashing scheme achieving one-access retrieval. *ACM Transactions on Database Systems*, 13(3):366–388, 1988.

[17] W. Litwin. Linear hashing: a new tool for file and table addressing. In *Proc. International Conference on Very Large Data Bases*, pages 212–223, 1980.

[18] H. G. Mairson. The effect of table expansion on the program complexity of perfect hash functions. *BIT*, 32(3):430–440, 1992.

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

[20] A. Pagh, R. Pagh, and M. Ružić. Linear Probing with 5-wise Independence. *SIAM Review*, 53(3):547–558, 2011.

[21] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51:122–144, 2004.

[22] H. Qi and C. U. Martel. Design and analysis of hashing algorithms with cache effects. Technical report, UC Davis, 1998.

[23] J. Schmidt, A. Siegel, and A. Srinivasan. Chernoff–Hoeffding Bounds for Applications with Limited Independence. *SIAM Journal on Discrete Mathematics*, 8:223, 1995.

[24] G. Tenenbaum. *Introduction to analytic and probabilistic number theory*. Cambridge Univ Press, 1995.

[25] E. Verbin and Q. Zhang. The limits of buffering: A tight lower bound for dynamic membership in the external memory model. In *Proc. ACM Symposium on Theory of Computing*, pages 447–456, 2010.

[26] M. Wegman and J. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.

[27] Z. Wei, K. Yi, and Q. Zhang. Dynamic external hashing: The limit of buffering. In *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, pages 253 – 259, 2009.