COMBINATORICA
Bolyai Society – Springer-Verlag

COMBINATORICA **18** (1) (1998) 121–132

NON-EXPANSIVE HASHING

NATHAN LINIAL* and ORI SASSON

Received February 5, 1996

In a *non-expansive* hashing scheme, similar inputs are stored in memory locations which are close. We develop a non-expansive hashing scheme wherein any set of size $O(R^{1-\varepsilon})$ from a large universe may be stored in a memory of size R (any $\varepsilon > 0$, and $R > R_0(\epsilon)$), and where retrieval takes $O(1)$ operations. We explain how to use non-expansive hashing schemes for efficient storage and retrieval of noisy data. A dynamic version of this hashing scheme is presented as well.

1. Introduction

A *dictionary* is a data structure for storing elements from universe U in memory M. It has to be capable of storing any subset $S \subseteq U$ whose cardinality is not too big, and an efficient implementation of the following operations is required:

- Membership queries of the form "Given $x \in U$ is $x \in S$? If it is, where is it stored in M?"
- Update operations: Add/Delete elements from S.

Consider the "noisy" version of the same problem : There is a notion of distance on U, i.e. it is a metric space, with metric d_U. There is some constant $\Delta > 0$ which is the measure of noise (or uncertainty), and on input x we are interested in retrieving every y such that $d_U(x,y) \leq \Delta$. In other words, the membership query is "given $x \in U$ decide whether there are any $y \in S$ such that $d_U(x,y) \leq \Delta$ and determine where these items are stored in M".

Of course, any dictionary may be adapted to deal with noisy data as well. Given $x \in U$, check for every $y \in U$ with $d_U(x,y) \leq \Delta$ whether it is stored in the dictionary. However, this simple procedure is applicable only if:

- For every $x \in U$, it is possible to quickly generate all $y \in U$ with $d_U(x,y) \leq \Delta$.
- Every memory location may be accessed at unit cost in each time.

Unfortunately, these assumptions fail in most interesting instances. The first assumption fails for real-time systems handling real world inputs (e.g. image

Mathematics Subject Classification (1991): 68R05, 68P05

* Part of the work was done while the author was visiting DIMACS.

recognition applications). The second assumption is not realistic when the memory M is very large, specifically when it is *paged*.

Are there storage/retrieval schemes that operate well regardless of these conditions? If U is mapped to M by functions that respect proximity in U, then on input $x \in U$, one would map x to $z \in M$ and search the vicinity of z. This concept is meaningful only if there is a metric d_M defined on M, which we henceforth assume.

Hashing is, of course, the most fundamental tool for dealing with the standard dictionary problem. It may seem, however, infeasible to rely on hashing if our goal is to respect proximity at U.[1]

This paper shows that this is a misconception. For the case where U and M are both (a long and a short) intervals with the usual metric, we construct small c-universal families of functions from U to M that do not expand distances.

Our method thus solves the noisy dictionary problem in a way that is nearly space-optimal and where retrieval time takes the optimal $O(\Delta)$ operations[2]. A dynamic version allows also updates at an optimal expected amortized cost.

The metric on U reflects some structure of the relevant inputs while d_M reflects the architecture of our storage space. Depending on the application domain U and the architecture used to deal with it M, there are numerous other instances of our general problem that would be very interesting.

Before we state our main result we need to resolve another technicality, namely what can be stored in a single memory location. To simplify matters, we assume that it holds a single element from U. This assumption is not essential and can be modified.

Definition 1. A function $h : U \to M$ is *non-expansive* (or *1-Lipschitz*) if for all $x, y \in U$, $d_M(h(x), h(y)) \leq d_U(x, y)$.

Definition 2. Consider a storage scheme that stores sets $S \subseteq U$ in memory M. Such a scheme is *c-non-expansive*, if for every $x \in U$ there is a set $B_1, \ldots, B_c$ of Δ-neighborhoods in M such that every $y \in S$ with $d_U(x, y) \leq \Delta$ is stored at some location in $\cup_{i=1}^{c} B_i$.

A storage scheme is *non-expansive* if it is c-non-expansive for some $c > 0$.

Theorem 3. *Let $U = [u]$, $\varepsilon > 0$, $M = [R]$, with $R > R_0(\varepsilon)$. U and M are equipped by the usual one-dimensional metric. There is a non-expansive hashing scheme that can store every $S \subseteq U$ with $|S| < R^{1-\varepsilon}$ in M. Retrieval is performed in a constant number of operations. The functions used for the hashing scheme are taken from a family of size $O(|U|)$.*

The following section surveys related work. Section 3 describes our novel family of non-expansive hashing functions, and describes a non-expansive hashing scheme

[1] e.g.: "... hash coding destroys any structure that the stored set may have, so it is a poor model of human memory." [7]

[2] It is still an intriguing open problem whether similar schemes can be devised where the number of operations depend on the *number* of elements $y \in S$ with $d_U(x, y) \leq \Delta$.

based on this family. We have also devised a dynamic version of our hashing scheme, and it is presented in the last section.

2. Related Work

The problem of hashing noisy data defined above is similar to the *Best Match* problem introduced by Minsky and Papert [8]. In their problem, D is a dictionary of strings over some alphabet Σ, and it is required to store D in such a way that on query string q, the member of D closest to q in Hamming distance be retrieved. The version where all members in D of distance d or less from q be retrieved is the "*Approximate Query (AQ)*" problem of [6].

Most of the work on these problems (see [9] and the references therein) focuses on deterministic schemes. However, deterministic hashing schemes are rather limited, especially for the dynamic dictionary problem (e.g. [2]), so we resort to randomized tools.

We were much inspired by the landmark paper by Fredman, Komlós and Szemerédi [5]. Their hashing scheme allows to store any subset of n elements from $U=[u]$ in memory of size $O(n)$. All queries are served in $O(1)$ operations.

The FKS hashing scheme is based on the notion of *pairwise independence* and so it conforms with the conventional view of hashing: Any two distinct elements in U are equally likely to be mapped to any two locations in M (regardless of their distance in U). Therefore, a naive adaptation of FKS to the noisy case will fail badly if the memory is *paged*. Our new hashing scheme described in the next section is based on a family of functions with a property weaker than pairwise independence.

3. Non-Expansive Hashing

3.1. The case $|S|=O(R^{\frac{1}{2}})$

We seek a family of non-expansive hash functions that can be used for perfect hashing.

Theorem 4. *There exists a family $\mathcal{H}$ of non-expansive functions from U to $[R]$, and a constant C ($C\leq 0.1$ will do) such that for any $S\subseteq U$ with $|S|\leq C\sqrt{R}$,*

$$Pr_{f\in\mathcal{H}}(f_{|S} \text{ is one-to-one}) \geq \frac{1}{2}$$

The family $\mathcal{H}$ contains $O(|U|)$ functions, each of which is computable in $O(1)$ operations.

Our idea is that the hash functions f that we use should have the property that for every $x\in U$ either $f(x+1)=f(x)+1$ or $f(x+1)=f(x)-1$. Obviously, this

condition implies that f is non expansive. Such a function f can be specified by selecting a "starting point" s, i.e. $f(0)=s$, and then defining $f(1)=s+1$, $f(2)=s+2$, and so on, until reaching a "turning point" t_0, where the direction is reversed. That is, if $f(a)=t_0-1$ and $f(a+1)=t_0$ then $f(a+2)=t_0-1, f(a+3)=t_0-2$ and so on, until the next "turning point" is reached, where again the direction is reversed. It is easy to see that for any $S\subseteq U$, $|S|=n$, there is a function h of this sort for which $h_{|S}$ is one-to-one. As stated, this approach is impractical since the size of this class of functions is exponential in n, and so members in this class do not have a short description. It is also unclear how to compute such functions efficiently.

We overcome these problems, by restricting ourselves to functions f whose turning points are not arbitrary but rather explicitly defined.

Note that a family of non-expansive mappings cannot possibly be pairwise independent. As we observe below it suffices to control the collision probabilities. That is, a *c-universal* family of hash functions can be used for the purpose of perfect hashing.

Definition 5. [1] A family $\mathcal{H}$ of functions from U to $[R]$ is *c-universal* if for every $x,y\in U$,

$$Pr_{f\in\mathcal{H}}(f(x)=f(y))\leq c\cdot\frac{1}{R}$$

We start with a variation on a lemma from FKS [5].

Lemma 6. *Let $\mathcal{H}$ be a c-universal family of functions from U to $[R]$. Then, for every $S\subseteq U$, with $|S|<\sqrt{\frac{R}{2c}}$*

$$Pr_{f\in\mathcal{H}}(f \text{ is one-to-one on } S)\geq\frac{1}{2}$$

Proof. Define a random variable X which counts for any function $f\in\mathcal{H}$ the number of pairs of elements in S that collide under f, i.e.:

$$X=\sum_{j\in M}\binom{|f^{-1}(j)|}{2}$$

Note that $X(f)=0$ iff $f_{|S}$ is one-to-one. Also, $X\geq 0$, and if the expectation $EX\leq\frac{1}{2}$, then at least half of the functions $f\in\mathcal{H}$ are one-to-one on S (by Markov inequality).

$$E(X)=\frac{1}{|\mathcal{H}|}\sum_{f\in\mathcal{H}}\sum_{j\in M}\binom{|f^{-1}(j)|}{2}=\frac{1}{|\mathcal{H}|}\sum_{x\neq y\in S}\sum_{f\in\mathcal{H}}\sum_{j\in M}1_{f(x)=f(y)=j}\leq$$

$$\leq|S|(|S|-1)\cdot\max_{x\neq y}Pr_{f\in\mathcal{H}}(f(x)=f(y))\leq c\frac{|S|(|S|-1)}{R}<\frac{1}{2}$$

∎

Let $L = \lfloor \hat{c}R \rfloor$ for some constant $\hat{c}$ (we pick $\hat{c} = \frac{1}{7}$). The "left" and "right" turning points used for our hash functions are always selected from the segments $[0, 2L-1]$ and $[R-2L+1, R]$, respectively. At "left" turning points we cease moving to the left and start moving right and the opposite for "right" turning points. A function f in our family $\mathcal{H}$ is specified as follows.

1. Choose a random "starting point" $f(0) = s_f \in [2L, R-2L]$
2. Let $\mathcal{P}$ be a family of *pairwise independent* functions from $[0, \frac{U}{R}]$ to $[0, L-1]$ (e.g. the hash functions from [5]). Randomly choose $\Phi \in \mathcal{P}$.
3. Consider the (unique) sequence $t_0, t_1, \ldots$ of integers from $[0, 2L-1]$ where

$$\sum_{i=0}^{j} t_i = jL + \Phi(j)$$

for every $j \geq 0$. We make our "left" turns at t_i for i odd and the "right" turns at $R - t_i$ for i even.

4. For any integer $0 \leq j < \frac{|U|}{R}$, let:

$$A_j = 2\left(jR - \sum_{i=0}^{2j-1} t_i\right) - s_f = 2(jR - (2j-1)L - \Phi(2j-1)) - s_f$$

$$B_j = 2\left((j+1)R - \sum_{i=0}^{2j} t_i\right) - s_f = 2((j+1)R - 2jL - \Phi(2j)) - s_f$$

5. Define f as follows:

$$f(x) = \begin{cases} x - A_0 & \text{if } 0 \leq 2x \leq A_0 + B_0 \\ x - A_j & \text{if } A_j + B_{j-1} < 2x \leq A_j + B_j \\ B_j - x & \text{if } A_j + B_j < 2x \leq A_{j+1} + B_j \end{cases}$$

Indeed, as stated above, $f(1) = s_f + 1$, $f(2) = s_f + 2$, and so on, until the first right "turning point" $R - t_0$ is reached and the direction is reversed. That is, if $f(a) = R - t_0 - 1$ and $f(a+1) = R - t_0$ then $f(a+2) = R - t_0 - 1$, and so on, until the next turn at t_1.

Equivalently, subsequent to the $2j$-th turn f is defined by $f(x) = x - A_j$ and $f(x) = B_j - x$ following the $(2j+1)$-st turn.

Define $\mathcal{H} = \{f : U \to [R] \mid \Phi \in \mathcal{P}, s_f \in [2L, R-2L]\}$

Given x, denote by $\tau_{x,f}$ the number of turns taken up to x by the function $f \in \mathcal{H}$.

Lemma 7. *For all $x \in U$, the set $\{\tau_{x,f} \mid f \in \mathcal{H}\}$ consists of at most four consecutive integers.*

Proof. Denote $\tau = \tau_{x,f}$, and fix any $x \in U$.

$$\tau = \begin{cases} 2j & \text{if } \ A_j + B_{j-1} < 2x \le A_j + B_j \\ 2j+1 & \text{if } \ A_j + B_j < 2x \le A_{j+1} + B_j \end{cases}$$

The above can be rewritten as:

$$\tau = \begin{cases} 2j & \text{if } 2j + \frac{3L - \Phi(2j-1) - \Phi(2j-2)}{R-2L} < \frac{x+s_f}{R-2L} \le (2j+1) + \frac{3L - \Phi(2j-1) - \Phi(2j)}{R-2L} \\ 2j+1 & \text{if } (2j+1) + \frac{3L - \Phi(2j-1) - \Phi(2j)}{R-2L} < \frac{x+s_f}{R-2L} \le 2(j+1) + \frac{3L - \Phi(2j+1) - \Phi(2j)}{R-2L} \end{cases}$$

In particular, $\tau = 2j$ is possible only when

$$2j + \frac{L}{R-2L} < \frac{x+s_f}{R-2L} \le 2j + 1 + \frac{5L}{R-2L}$$

and similarly $\tau = 2j+1$ is possible only when

$$2j + 1 + \frac{L}{R-2L} < \frac{x+s_f}{R-2L} \le 2j + 2 + \frac{5L}{R-2L}$$

As long as $L < \frac{R}{6}$, the "even" intervals $\{[2j + \frac{L}{R-2L}, 2j+1+\frac{5L}{R-2L}]\}_{j \ge 0}$ are disjoint, similarly for "odd" intervals $\{[2j+1+\frac{L}{R-2L}, 2j+2+\frac{5L}{R-2L}]\}_{j \ge 0}$.

Also, the values of $\frac{x+s_f}{R-2L}$ span an interval of length 1. This interval may intersect with at most two odd intervals and at most two even intervals, because the intervals are separated and their left ends increase by 1. Consequently, for every $x \in U$, there are at most four numbers in $\{\tau_{x,f} \mid f \in \mathcal{H}\}$. ∎

Lemma 8. *$\mathcal{H}$ is a family of c-universal hash functions.*

Proof. Fix any $x > y$ in U. Given $\tau_{x,f}$ and $\tau_{y,f}$, the condition $f(x) = f(y)$ is equivalent to an equation involving two different values of Φ. The equation depends on the parities of $\tau_{x,f}$ and $\tau_{y,f}$.

For example, if $\tau_{x,f}$ is even, and $\tau_{y,f}$ is even, then:

$$f(x) = x - A_{\frac{\tau_{x,f}}{2}}$$

and

$$f(y) = y - A_{\frac{\tau_{x,f}}{2}}$$

Hence we obtain:

$$x - y = (\tau_{x,f} - \tau_{y,f})(R-2L) - 2(\Phi(\tau_{x,f} - 1) - \Phi(\tau_{y,f} - 1))$$

Similar equations hold for the other 3 cases.

The pairwise independence of $\mathcal{P}$ implies that such an equality holds with probability $O(\frac{1}{R})$. In other words, given β, and any $u, v \in U$,

$$Pr_{\Phi\in\mathcal{P}}(\Phi(u)-\Phi(v)=\beta) \leq \sum_{\alpha=1}^{L} Pr_{\Phi\in\mathcal{P}}(\Phi(u)=\alpha+\beta \mid \Phi(v)=\alpha) Pr_{\Phi\in\mathcal{P}}(\Phi(v)=\alpha)$$

But $\mathcal{P}$ is pairwise independent so each of the $O(R)$ terms in the sum is $O(\frac{1}{R^2})$, hence

$$Pr_{\Phi\in\mathcal{P}}(\Phi(u)-\Phi(v)=\beta) \leq O\left(\frac{1}{R}\right)$$

Let $\mathcal{T}_x = \{\tau_{x,f} \mid f \in \mathcal{H}\}$ and $\mathcal{T}_y = \{\tau_{y,f} \mid f \in \mathcal{H}\}$. Then,

Hence we obtain:

$$Pr_{f\in\mathcal{H}}(f(x)=f(y)) =$$

$$\sum_{s\in\mathcal{T}_x}\sum_{s\in\mathcal{T}_y} Pr(f(x)=f(y) \mid \tau_{x,f}=s \text{ and } \tau_{y,f}=t) \leq |\mathcal{T}_x| \cdot |\mathcal{T}_y| \cdot O\left(\frac{1}{R}\right) \leq O\left(\frac{1}{R}\right)$$

Since $|\mathcal{T}_x|, |\mathcal{T}_y| = O(1)$ by lemma 7. ∎

3.2. The case $|S| \leq O(R^{1-\varepsilon})$

In this section we present a hashing scheme for storing a larger set S with $|S| < O(R^{1-\varepsilon})$ (any $\varepsilon > 0$ and $R > R_0(\varepsilon)$). This hashing scheme allows serving a query in the noisy dictionary problem in $O(1)$ operations.

At this point we must further deviate from the FKS scheme. Their two-layer method becomes inapplicable, since the second layer will destroy the non-expansive property achieved for $|S| \leq O(\sqrt{R})$. We therefore resort to an alternative idea.

Let $\mathcal{H}$ be a family of c-universal non-expansive hash functions, and $t = O(\log \frac{1}{\varepsilon})$ a constant.

S will be hashed into t separate tables of size R each. Select $h_1 \in \mathcal{H}$ at random, and hash the elements of S into the first table. The expected number of pairs $x, y \in S$ that collide under h_1 is less than $\frac{c|S|^2}{R}$ (by the proof of lemma 6). Consequently, the expected number of elements $x \in S$ that collide with another element $y \in S$ is at most $\frac{2c|S|^2}{R}$ (in the extremal case elements collide in pairs). Hence most possible selections of h_1 have the property that the actual number of elements $x \in S$ that collide with another element $y \in S$ is less than $\frac{2c|S|^2}{R}$. Select such h_1 and store the

elements that do not collide under it in the first table. The colliding elements are then hashed using a randomly selected $h_2 \in \mathcal{H}$ into the second table. The expected number of elements which are hashed to the same location in the second table is less than $\frac{8c^3|S|^4}{R^3}$. Continue this process inductively. For most selections of h_{t-1}, fewer than $\frac{(2c)^{2^{t-1}-1}|S|^{2^{t-1}}}{R^{2^{t-1}-1}}$ elements are expected to collide under h_{t-1}. For $t = O(\log \frac{1}{\varepsilon})$ this size is less than $\sqrt{\frac{R}{2c}}$.

Assuming that at most $\sqrt{\frac{R}{2c}}$ collide under h_{t-1}, most selections of h_t will guarantee perfect (i.e. one-to-one) hashing into the last table.

For a membership query "given $x \in U$ decide whether there are any $y \in S$ such that $d_U(x,y) \leq \Delta$ and determine where these items are stored in M", we probe the Δ-neighborhood of $h_i(x)$ in the i-th hash table, for each $1 \leq i \leq t$.

4. The Dynamic Case

In the spirit of [2], this section describes a dynamic version of the non-expansive hashing scheme. This version achieves expected amortized complexity $O(1)$ for any sequence of updates.

First, we consider the more relaxed problem where memory size is allowed to be quadratic in the size of the stored set.

4.1. The case of quadratic storage space

A dynamic storage scheme is required to process a sequence of insertions and deletions. Occasionally, we will *rehash* i.e. select a new set of hash functions. As the current set S of stored elements grows and shrinks we may also *resize* our memory to an either larger or smaller size.

Let $\mathcal{H}$ be a family of c-universal non-expansive hash functions, as in Section 3.

Our policy for resizing the memory, i.e. changing the size of the table R is as follows:

- Following an insert operation, if $|S|^2 \geq \frac{R}{2c}$ then enlarge the table to $R = 3c|S|^2$.
- Following a delete operation, if $|S|^2 < \frac{R}{4c}$ then set $R = \frac{8c}{3}|S|^2$.

Specifically, the following inequality is maintained at all times:

$$\frac{1}{4} \leq \frac{c|S|^2}{R} < \frac{1}{2}$$

We use one hash function $h \in \mathcal{H}$, which is one-to-one on the current S. If the next element to be inserted, x, is such that location $h(x)$ is already occupied, we rehash.

Our analysis goes as follows: Fix the (initially unknown) sequence of update requests σ. This sequence is split into *intervals* that end at those locations in σ where our rule calls for resizing. Note that the times at which we resize depend on σ, and not on the steps taken by the storage and retrieval scheme. Specifically, it is independent of the random choices made by the algorithm. Note also that our rule for resizing is such that at the two ends of any interval (i.e., at two consecutive resize operations), the numbers of currently stored items, differ by an absolute constant factor. Fix some interval, I, and let n be the smaller of these two numbers. Also, there are $\Omega(n)$ requests for update throughout I, so it is possible to further break I to *segments* of length $\Theta(n)$ requests each. Our analysis will show that the expected number of operations for each such segment is only $O(n)$. The resizing that takes place at the beginning of I costs only $O(n)$ operations, which we charge to the first segment in I. As long as no collisions occur, every request is served at an $O(1)$ cost.

The analysis of rehashing costs is a little more involved: We claim that only a constant number of rehash steps are expected to take place within each segment. To see this, consider a segment at the beginning of which there are, say, n items stored in memory. Observe that only $O(n)$ items are added to M throughout the interval. We can even afford to ignore the fact that some of them are also deleted within the same period. Let T, with $|T| \leq a \cdot n$, (where a is an absolute constant) be the set of these (initially unknown) elements. We have picked our parameters so that for every set $T \subseteq U$ of cardinality $a \cdot n$, most hash functions that we select are one-to-one on T (following the calculation in the proof of lemma 6). If this is the case, then *no* rehashing will be required throughout this segment. Since our chance of success is bounded away from zero, the expected number of selections for $h \in \mathcal{H}$ within the segment, and hence the expected number of rehashings, is only $O(1)$, as claimed.

4.2. The case of slightly super-linear storage space

This general case can be treated similarly to the case of quadratic storage. One modification is that $t = O(\log \frac{1}{\varepsilon})$ tables are used, each of which has size tR. Another difference is that resizing is geared towards maintaining the inequality

$$\frac{1}{4} \leq \frac{c|S|^{1+\epsilon}}{R} < \frac{1}{2}$$

at all times. Namely, the policy used for resizing the memory is:

- Following an insert operation, if $|S|^{1+\epsilon} \geq \frac{R}{2c}$ then enlarge the table to $R = 3c|S|^{1+\epsilon}$.
- Following a delete operation, if $|S|^{1+\epsilon} < \frac{R}{4c}$ then set $R = \frac{8c}{3}|S|^{1+\epsilon}$.

We keep a set of t hash functions h_i. When a new element x is to be inserted into the table, we store it in location $h_i(x)$ of the i-th table, for the smallest $0 \leq i < t$ such that this location is not occupied. If all locations $h_i(x)$ are occupied, then we

rehash. Queries and deletions for an element $x \in U$ are handled by checking all locations $h_i(x)$.

The analysis used above to show $O(1)$ expected cost of operations is still valid, only the analysis of rehashing costs is slightly different. Consider again a set $T \subseteq U$ of items that appear in the update requests within one segment. Previously, we only demanded that $h \in \mathcal{H}$, used for storage throughout this interval be one-to-one on T with probability bounded away from zero (over all possible choices of h), since the selections of the h_i's are independent, and since $(1-\frac{1}{t})^t \geq \Omega(1)$.

This condition needs to be strengthened now so that our probability of success be at least $1-\frac{1}{t}$ for each h_i. Hence, for a tuple $(h_i)_{i=1}^t$ the probability that it can be used for storage throughout the interval will be bounded away from zero.

Denote by $Q_1(T)$ the number of elements of T which collide when hashing T with h_1. Define inductively the set $Q_j(T)$ of elements of T which collide when hashing $Q_{j-1}(T)$ with the function h_j.

Claim 9.

$$Pr_{(h_i)_{i=1}^t \in \mathcal{H}^t}\left(|Q_{t-1}(T)| < \sqrt{\frac{R}{2c}}\right) \geq \left(1-\frac{1}{t}\right)^{t-1}$$

Proof. In the following proof, all probabilities are computed in the space $\mathcal{H}^t$.

$$Pr\left(|Q_{t-1}(T)| < \sqrt{\frac{R}{2c}}\right) =$$

$$Pr\left(|Q_{t-1}(T)| < \sqrt{\frac{R}{2c}} \;\middle|\; |Q_{t-2}(T)| < \frac{(2c)^{2^{t-2}-1}|T|^{2^{t-2}}}{R^{2^{t-2}-1}}\right) \cdot$$

$$Pr\left(|Q_{t-2}(T)| < \frac{(2c)^{2^{t-2}-1}|T|^{2^{t-2}}}{R^{2^{t-2}-1}}\right)$$

Continuing this conditioning inductively, we get

$$Pr\left(|Q_{t-1}(T)| < \sqrt{\frac{R}{2c}}\right) \geq Pr\left(Q_1(T) < \frac{2c|T|^2}{R}\right) \cdot$$

$$\prod_{l=2}^{t-1} Pr\left(|Q_l(T)| < \frac{(2c)^{2^l-1}|T|^{2^l}}{R^{2^l-1}} \;\middle|\; |Q_{l-1}(T)| < \frac{(2c)^{2^{l-1}-1}|T|^{2^{l-1}}}{R^{2^{l-1}-1}}\right)$$

since $t = O(\log \frac{1}{\varepsilon})$ implies

$$\frac{(2c)^{2^{t-1}-1}|T|^{2^{t-1}}}{R^{2^{t-1}-1}} \leq \sqrt{\frac{R}{2c}}$$

Now, the expected number of elements with collisions under h_l is at most $\frac{2c|Q_{l-1}(T)|^2}{tR}$. Hence, given that

$$|Q_{l-1}(T)| < \frac{(2c)^{2^{l-1}-1}|T|^{2^{l-1}}}{R^{2^{l-1}-1}}$$

The expected size of the set $Q_l(T)$ is less than $\frac{(2c)^{2^l-1}|T|^{2^l}}{tR^{2^l-1}}$.

Therefore by Markov inequality,

$$Pr\left(|Q_l(T)| > \frac{(2c)^{2^l-1}|T|^{2^l}}{R^{2^l-1}} \;\middle|\; |Q_{l-1}(T)| < \frac{(2c)^{2^{l-1}-1}|T|^{2^{l-1}}}{R^{2^{l-1}-1}}\right) \leq \frac{1}{t}$$

And so,

$$Pr\left(|Q_l(T)| < \frac{(2c)^{2^l-1}|T|^{2^l}}{R^{2^l-1}} \;\middle|\; |Q_{l-1}(T)| < \frac{(2c)^{2^{l-1}-1}|T|^{2^{l-1}}}{R^{2^{l-1}-1}}\right) \geq 1 - \frac{1}{t}$$

Hence we obtain

$$Pr\left(|Q_t(T)| < \frac{\sqrt{R}}{2}\right) \geq \left(1 - \frac{1}{t}\right)^{t-1}$$ ∎

Corollary 10.

$$Pr_{(h_i)_{i=1}^t \in \mathcal{H}^t}(T \text{ is hashed perfectly into the } t \text{ hash tables}) > \frac{1}{e}$$

(Here $e = 2.71828\ldots$ is Euler's constant.)

Proof. Using a similar argument to the one used in the previous proof,

$$Pr\left(Q_t(T) < 1 \;\middle|\; Q_{t-1} < \sqrt{\frac{R}{2c}}\right) \geq 1 - \frac{1}{t}$$

hence

$$Pr(Q_t(T) < 1) \geq \left(1 - \frac{1}{t}\right)^t > \frac{1}{e}$$

And since this probability (which is the probability of a tuple being suitable for perfectly hashing T into the set of tables) is bounded away from zero, the expected number of rehashings in a segment (using the terminology of the previous subsection) is $\leq e$. ∎

The following theorem states our final result.

Theorem 11. *For any $\delta > 0$ there exists a dynamic non-expansive hashing scheme that requires only $O(n^{1+\delta})$ space to store n elements. Queries are served in $O(1)$ operations, and a sequence of updates is performed with expected amortized cost of $O(1)$ operations.*

References

[1] J. L. Carter and M. N. Wegman,: *Universal Classes of Hash Functions*, Proceedings of the 9th ACM Symposium on Theory of Computing, 1977, 106–112.

[2] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan: *Dynamic Perfect Hashing: Upper and Lower Bounds*, Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, 1988, 524–531.

[3] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas: *Neighborhood preserving hashing and approximate queries*, 5th ACM Symposium on Discrete Algorithms, 1994.

[4] D. Dolev, Y. Harari, and M. Parnas: *Finding the neighborhood of a query in a dictionary*, 2nd Israeli Symposium on Theory of Computing and Systems.

[5] M. L. Fredman, J. Komlós, and E. Szemerédi: *Storing a Sparse Table with $O(1)$ Worst Case Access Time*, Journal of the ACM, Vol. 31, No. 3, July 1984, 538–544.

[6] D. Greene, M. Parnas, and F. Yao: *Multi-Index Hashing for Information Retrieval*, Proceedings of 35th Annual IEEE Symposium on Foundations of Computer Science, 1994.

[7] P. Kanerva: *Sparse Distributed Memory*, MIT Press, Cambridge, Massachusetts, 1988, 12.

[8] M. Minsky and S. Papert: *Perceptrons*, MIT Press, Cambridge, Massachusetts, 1969, 222–225.

[9] M. Parnas: *Robust Algorithms and Data Structures for Information Retrieval*, Ph. D. dissertation, Hebrew University, Jerusalem, Israel, 1994.

Nathan Linial

Hebrew University,
Jerusalem, Israel.
nati@cs.huji.ac.il

Ori Sasson

Hebrew University,
Jerusalem, Israel.
ori@cs.huji.ac.il