Non-uniform attacks against one-way functions and PRGs

Anindya De[*] Luca Trevisan[†] Madhur Tulsiani[‡]

November 8, 2009

Abstract

We study the power of non-uniform attacks against one-way functions and pseudorandom generators.

Fiat and Naor [FN99] show that for every function $f : [N] \rightarrow [N]$ there is an algorithm that inverts f everywhere using (ignoring lower order factors) time, space and advice at most $N^{3/4}$.

We show that an algorithm using time, space and advice at most

$$\max\{\epsilon^{\frac{5}{4}} N^{\frac{3}{4}} \ , \ \sqrt{\epsilon N}\}$$

exists that inverts f on at least an ϵ fraction of inputs. A lower bound of $\tilde{\Omega}(\sqrt{\epsilon N})$ also holds, making our result tight in the "low end" of $\epsilon \leq \sqrt[3]{\frac{1}{N}}$.

(Both the results of Fiat and Naor and ours are formulated as more general trade-offs between the time and the space and advice length of the algorithm. The results quoted above correspond to the interesting special case in which time equals space and advice length.)

We also show that for every length-increasing generator $G : [N] \rightarrow [2N]$ there is a algorithm that achieves distinguishing probability ϵ between the output of G and the uniform distribution and that can be implemented in polynomial (in $\log N$) time and with advice and space $O(\epsilon^2 \cdot N \log N)$. Alternatively, it can be implemented as a circuit of size $O(\epsilon^2 \cdot N)$. We prove a lower bound of $S \cdot T \geq \Omega(\epsilon^2 N)$ where T is the time used by the algorithm and S is the amount of advice.

We prove stronger lower bounds in the *common random string* model, for families of one-way permutations and of pseudorandom generators.

Keywords: One-way functions, pseudorandom generators, random permutations, time-space trade-offs

[*]Computer Science Division, University of California, Berkeley, CA, USA. anindya@cs.berkeley.edu. Supported by the "Berkeley Fellowship for Graduate Study" and US-Israel BSF grant 2006060.

[†]Computer Science Division, University of California, Berkeley, CA, USA. luca@cs.berkeley.edu. This material is based upon work supported by the National Science Foundation under grant No. CCF-0729137 and by the BSF under grant 2006060.

[‡]Institute for Advanced Study, Princeton, NJ, USA. madhurt@math.ias.edu. This material is based upon work supported by the National Science Foundation under grant No. CCF-0832797 and IAS Sub-contract no. 00001583. Work done partly when the author was a graduate student at UC Berkeley.

ISSN 1433-8092

1 Introduction

In the applied cryptography literature, a cryptographic primitive with a key of length k is typically considered "broken" if the key can be recovered in time less than 2^k, that is, faster than via an exhaustive brute force search. Implicit in this attitude is the belief in the existence of primitives for which a brute force attack is optimal. A time t brute force attack against a one-way function $f : \{0,1\}^n \to \{0,1\}^n$, consisting in trying about t random guesses for the inverse, only succeeds with probability about $t/2^n$, and a brute force attack that attempts to distinguish a length increasing generator $G : \{0,1\}^{n-1} \to \{0,1\}^n$ from the uniform distribution by attempting to guess the seed achieves distinguishing probability about $t/2^n$. Is it plausible that such trade-offs are optimal? Would it be plausible to assume that AES with 128 key bit cannot be distinguished from a random permutation with distinguishing probability more than 2^{-40} by adversaries running in time 2^{60}?[1]

If we apply a non-uniform measure of complexity, that is, if we restrict ourselves to a fixed finite one-way function or pseudorandom generator, and allow our adversary to use precomputed information as advice, then it turns out that the above "brute force" bounds can always be improved upon.

In 1980, Hellman [Hel80] proved that for every one-way *permutation* $f : [N] \to [N]$ (for this discussion, it will be convenient to set $N = 2^n$ and identify $\{0,1\}^n$ with $[N]$) and for every parameters S, T satisfying $S \cdot T \geq N$, there is a data structure of size $\tilde{O}(S)$ and an algorithm that, with the help of the data structure, given $f(x)$ is always able to find x in time $\tilde{O}(T)$. (The notation $\tilde{O}(\cdot)$ hides lower order factors that are polynomial in $\log N$; we will ignore such factors from now on in the interest of readability.) We shall refer to S, the size of the pre-computed data structure used by the algorithm, as the *space* used by the algorithm.

In particular, every one-way permutation can be inverted in time $\sqrt{N}$ using $\sqrt{N}$ bits of advice.[2]

Hellman algorithm only requires oracle access to the permutation. Yao [Yao90] proves that, in this oracle setting, Hellman's trade-off is tight for random permutations. (See also [Gol09].)

Hellman also considers the problem of inverting a random function $f : [N] \to [N]$ given oracle access to f. He provides a heuristic argument suggesting that for every S, T satisfying $TS^2 \geq N^2$, and with high probability over the choice or a random function $f : [N] \to [N]$, there is a data structure of size S and an algorithm of complexity T that inverts f everywhere using the data structure and given oracle access f. This trade-off yields the interesting special case $S = T = N^{2/3}$.

Fiat and Naor [FN99] prove Hellman's result rigorously, and are able to handle arbitrary functions, not just random functions. If the given function $f : [N] \to [N]$ has collision probability[3] λ, then the algorithm of Fiat and Naor requires the trade-off $TS^2 \geq \lambda \cdot N^3$. Note that with high probability a random function has collision probability about $1/N$ (recall that we ignore $(\log N)^{O(1)}$ terms), and so one recovers Hellman's tradeoff. For general functions, Fiat and Naor are able to prove the trade-off $TS^3 \geq N^3$, which has the special case $S = T = N^{3/4}$.

[1]The answer to the last question is no. It follows from our results in Appendix 9 that there is a distinguisher that makes two queries, then performs a computation realizable as a circuit of size 2^{56}, assuming a complete basis of fan-in two gates, and achieves distinguishing probability $\geq 2^{-40}$ between AES_{128} and a random permutation $\{0,1\}^{128} \to \{0,1\}^{128}$. Otherwise, after the two oracle queries, the distinguisher can be implemented in a 64-bit architecture with two table look-ups and three unit-cost RAM operations, given access to a precomputed table of 2^{49} entries.

[2]This doesn't mean that there is a circuit of size $\tilde{O}(\sqrt{N})$; the running time of $\tilde{O}(\sqrt{N})$ is in the RAM model. The relationship between non-uniform time/space complexity measures and circuit complexity is the following: a circuit of size C can be simulated using time at most $\tilde{O}(C)$ given a pre-computed data structure of size $\tilde{O}(C)$; and an algorithm that uses time T and a pre-computed data structure of size S can be simulated by a circuit of size $\tilde{O}((S+T)^2)$.

[3]Here by the collision probability of a function we mean the probability that after sampling two independent random inputs x, y we have $f(x) = f(y)$.

Barkan, Biham, and Shamir [BBS06] prove that the $TS^2 = N^2$ trade-off of Fiat and Naor for random functions is optimal under certain assumptions on what is stored in the data structure and on the behavior of the algorithm.

The result of Fiat and Naor can also be applied to the task of distinguishing a given pseudorandom generator from the uniform distribution (and hence a given pseudorandom permutation from a random permutation or a given pseudorandom function from a random function) by recovering the seed. We are not aware of previous work that focused specifically on the complexity of distinguishers for pseudorandom generators. Two related results, however, should be mentioned. It has been known for a long time (going back to, as far we know, [AGHP92]) that every distribution that has constant statistical distance from the uniform distribution, and, in particular, the output of any length increasing generator, can be distinguished from the uniform distribution over n bits using a parity function (of linear circuit complexity), and with distinguishing probability $\Omega(2^{-\frac{n}{2}})$. The other result is due to Andreev, Clementi and Rolim [ACR97], who prove that for every boolean predicate $P : \{0,1\}^n \rightarrow \{0,1\}$ and every ϵ there is a circuit of size $O(\epsilon^2 2^n)$ that computes P on at least a $1/2 + \epsilon$ fraction of inputs. This implies that for every pseudorandom generator of the form $x \rightarrow f(x)P(x)$, where f is a permutation and P is a hard-core predicate for f, and every $\epsilon > 0$, there is a circuit of size $O(\epsilon^2 2^n)$ that achieves distinguishing probability ϵ.

Our Results

We show that for every $f : [N] \rightarrow [N]$ and every ϵ there is an algorithm that inverts f on an ϵ fraction of inputs and whose time complexity, space complexity and advice length are bounded by

$$\tilde{O}\left(\max\left\{\sqrt{\epsilon N}\ , \epsilon^{\frac{5}{4}} N^{\frac{3}{4}}\right\}\right)$$

Here the $\tilde{O}$ hides factors of $2^{\text{poly}\log\log}$. It follows from known results, and we present a proof in Appendix 10, that, in an oracle setting, it is not possible to do better than $\Omega(\sqrt{\epsilon N})$, so our result is best possible when $\epsilon < N^{1/3}$.

Indeed, we establish the following more general trade-off: for every $T < 1/\epsilon$ and S satisfying the trade-off $ST = \epsilon N$ we can construct an algorithm that has time T and uses a data structure of size S (up to lower order factors); for every $T > 1/\epsilon$, we can use time T and space S provided $TS^3 = \epsilon^5 N^3$. As we discuss below, a straight-forward application of the analysis of Fiat and Naor would have given a trade-off $TS^3 = \epsilon^3 N^3$, or a time and space complexity $\tilde{O}(\epsilon^{\frac{3}{4}} N^{\frac{3}{4}})$ in the $T = S$ case. For comparison, when $\epsilon = N^{-1/3}$, we can achieve (optimal) time and space $N^{1/3}$; the straight-forward use of the Fiat-Naor analysis would have given time and space $\sqrt{N}$. Given an upper bound λ on the collision probability, we can achieve the optimal trade-off $TS = \epsilon N$ if $S \geq \epsilon^2 N^2 \lambda$, and the trade-off $TS = \epsilon^2 N^2 \lambda$ otherwise. For example, if we have a function with collision probability close to $1/N$, and we want to achieve inversion probability $\epsilon = N^{-1/4}$, then we can do so, using the latter construction, employing time, space and advice at most $N^{5/12} = N^{.416\dots}$; using our generic construction (which applies to functions of arbitrary collision probability) would have given a complexity of $N^{7/16} = N^{.4375}$. There is a small catch : A technicality in the construction requires $S = \tilde{\Omega}(\sqrt{\epsilon N})$ to get the aforementioned trade-off curves. In fact, all the positive results (past results as well as ours) that we state in this paper regarding time-space trade-offs require $S = \tilde{\Omega}(\sqrt{\epsilon N})$

The difference between our analysis and the one in [FN99] is explained in Section 2.1 below.

Given an arbitrary length-increasing generator $G : \{0,1\}^n \rightarrow \{0,1\}^m$, $m > n$, we show that, for every ϵ, there is a distinguisher that runs in polynomial time, uses a data structure of size $\O(\epsilon^2 2^n)$, and achieves distinguished probability ϵ. The distinguisher can also be implemented as a circuit of size $O(\epsilon 2^n)$. Notably, the distinguisher need not have oracle access to G, and so our result applies to generators constructed for applications in derandomization, in which the generator may have complexity $2^{O(n)}$, or even higher. In this setting, in which the complexity of the generator is not bounded, it is easy to see that advice $\Omega(\epsilon^2 2^n)$ is necessary. We also present a simpler construction that achieves the slightly worse circuit size $O(\epsilon^2 n 2^n)$.

Open Questions

It remains open to either improve the Fiat-Naor construction or to prove a stronger lower bound for the problem of inverting a random function or an arbitrary function everywhere. It is plausible that the optimal trade-off $ST = N$, while achievable for permutations, is impossible to achieve for general functions, maybe even impossible for random functions. Such a separation between the complexity of dealing with general or random functions versus permutations would be extremely interesting.

If one wants to invert a random permutation or function uniformly (that is, given no advice), then the lower bound $T \geq N$ (ignoring lower-order factors) holds. A quantum computer, however, can achieve $T = \sqrt{N}$ [Gro96], which is optimal [BBBV97]. What is the complexity of inverting a random permutation, a random function, or an arbitrary function with a quantum computation that takes advice?

We do not have matching upper and lower bounds for the problem of constructing distinguishers for pseudorandom generators, except in the extremal case $T = 1$, $S = \epsilon^2 N$. Is $T = \epsilon^2 N$, $S = 1$ achievable? More generally, for what range of parameters is it possible to achieve distinguishability even though inversion of one-way permutations or functions is impossible?

2 Overview of Previous Techniques and Our Results

2.1 One-Way Functions

How can one invert one-way functions, in general, faster than by brute force?

2.1.1 An Overview of the Ideas of Hellman and of Fiat and Naor

If we are given a one-way *permutation* $f : [N] \to [N]$, then it is easy to construct an inverter for $f()$ that uses time and space $\tilde{O}(\sqrt{N})$. Suppose for simplicity that $f()$ is a cyclic permutation and that $N = s^2$ is a perfect square: then pick $\sqrt{N}$ "equally spaced" points $x_1, \ldots, x_s$, such that $x_{i+1} = f^{(s)}(x_i)$, and create a data structure to store the pairs (x_i, x_{i+1}). Then given y, we compute $f(y)$, $f(f(y))$, and so on, until, for some j, we reach a point $f^{(j)}(x)$ which is one of the special points in the data structure. Then we can read from the data structure the value $f^{(j-s)}(y)$, and then by repeatedly computing f again we will eventually reach $f^{(-1)}(y)$. Note that this takes $O(s)$ evaluations of f and table look-ups, so both the time and space complexity are approximately $s = \sqrt{N}$. If $f()$ is not cyclic, we do a similar construction for each cycle of length less than s, and if N is not a perfect square we can round s to $\lceil \sqrt{N} \rceil$.

Abstractly, this construction works for the following reason. Consider the graph $G_f = ([N], E)$ that has $[N]$ as set of vertices and that for every x has the directed edge $(x, f(x))$. Then, if f is a permutation, it is possible to cover G_f using $\sqrt{N}$ edge-disjoing paths, each of length $\sqrt{N}$ or, more

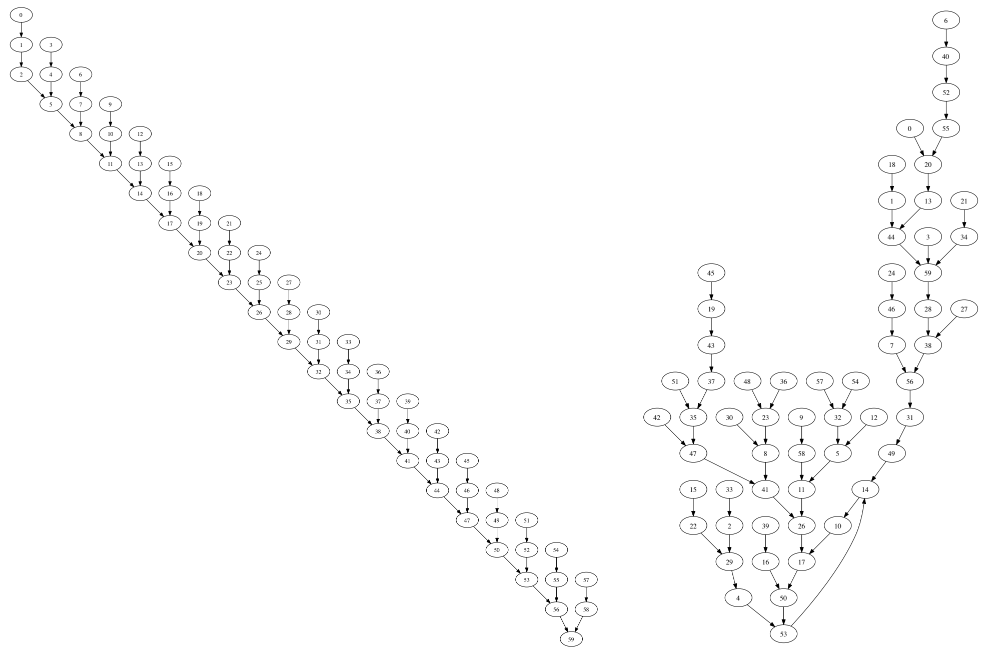

Figure 1: A graph G_f that cannot be partitioned into few edge-disjoing paths and the graph $G_{f\circ g}$ where g is a random permutation.

general, S edge-disjoing paths of length T, provided $ST \geq N$. Furthermore, if f is a function such that G_f can be covered using S edge-disjoint paths, each of length at most T, then we have an algorithm to invert f using space S and time T.

The problem is that, in general, no good collection of paths may exist. Suppose, for example, that G_f looks like the graph on the left in Figure 1: a directed path of length $\frac{1}{3}N$ with a length-2 path joining in at each point. Then we see that there is a set S (the vertices of indegree zero in the picture) of size $N/3$ such that no path can contain more than one vertex of S, and so no collection of $o(N)$ paths can cover the entire graph.

Hellman [Hel80] considers the case in which $f()$ is a *random* function. Then, even though it's not clear how many edge-disjoint paths of what length can cover G_f it is not hard to see that one can find $N^{\frac{1}{3}}$ paths of length $N^{\frac{1}{3}}$ having very few "collisions." This gives a construction that uses time and space $N^{\frac{1}{3}}$ and that inverts $f()$ at $N^{\frac{2}{3}}$ points. Hellman then suggests to modify $f()$ by composing it with a fixed permutation of the input bits, and to reason heuristically as if the new function behaved as an independently chosen new random function. Then one can repeat the construction, and have a new algorithm of time and space complexity $N^{\frac{1}{3}}$ that inverts $f()$ at $N^{\frac{2}{3}}$ points, which are assumed to be an independent random subset of size $N^{\frac{2}{3}}$. After iterating this process $N^{\frac{1}{3}}$ times one has $N^{\frac{1}{3}}$ candidate algorithms, each of time and space complexity $N^{\frac{1}{3}}$, such that, for every x, $f(x)$ is inverted by at least one of the algorithms. Overall, one gets an algorithm of complexity $N^{\frac{2}{3}}$ that inverts f everywhere.

Fiat and Naor [FN99] make Hellman's argument rigorous. The idea of Fiat and Naor is to pick a good random hash function g, and then work with the new function $h(x) := f(g(x))$. (See Figure 1 for an example of the effect of this randomization.) If g were a truly random function, and f where a function such that every output has few pre-images, then one can repeat Hellman's calculation that $N^{\frac{1}{3}}$ nearly disjoint paths of length $N^{\frac{1}{3}}$ exist. Picking $N^{\frac{1}{3}}$ random functions g_i then would give a rigorous version of the full argument, except for the dependancy on several random oracles. For a more general trade-off, it is possible to pick m nearly disjoint paths of length t provided that $m \cdot t^2 < N$, and then iterate the construction r times, where $r = N/mt$. Thus one gets a data structure of size $r \cdot m$, plus the space needed to store the descriptions of the hash functions, and an inversion whose complexity is dominated by the complexity of evaluating the r random hash functions at t points each. Fiat and Naor then show that each g_i only needs to be k-wise independent where k is approximately t, the length of the paths. While one evaluation of a t-wise independent hash function would take time t, Fiat and Naor show that the overall time for the rt evaluation can be made $t^2 + rt$ via a careful evaluation process and amortized analysis. The different g_i, in turn, only need to be pair-wise independent with respect to each other. Overall, the r hash functions can be represented using only about t bits, so that the space complexity is of the order of $r \cdot m + t$. Choosing the parameters r, m, t optimally shows that the time-space tradeoff $TS^2 = N^2$ is achievable.

For general functions, the above ideas continue to work if the collision probability λ of the distribution $f(U_{[N]})$ is small. In particular, one can have an algorithm of space $S = m \cdot r + t$ and time $T = t^2 + t \cdot r$ provided that $m \cdot t^2 \leq 1/\lambda$ and $m \cdot t \cdot r \geq N$. This optimizes to the time-space tradeoff $TS^2 = \lambda \cdot N^3$.

For functions having large collision probability, the idea is to create an additional look-up table L (we also refer to it as a list), containing, for each of the ℓ elements y such that $f^{(-1)}(y)$ is largest, the pair (x, y) where x is an arbitrary preimage of y. Then, given $f(x) \in L$ we can immediately find an inverse by searching L, and the problem of inverting f reduces to the problem of inverting the restriction of f to $\{0,1\}^n - f^{(-1)}(L)$, which, intuitively, is the problem of inverting a function of low collision probability. More precisely, if we define the "effective" collision probability of f relative to L as the probability that, picking x, x' uniformly at random we have $f(x) = f(x')$ conditioned on $f(x) \notin L$, then the effective collision probability is at most $1/\ell$. The $TS^2 = \lambda N^3$ trade-off can be extended to the case in which λ is the effective collision probability, although at the additional cost of ℓ in the space. The optimal choice ends up being $\ell = S$, and so the trade-off becomes $TS^3 = N^3$. One additional difficulty that comes up in the analysis is that we need hash functions g_i with the property that $g_i(f(x)) \notin f^{(-1)}(L)$ if $f(x) \notin L$. This is achieved by realizing g_i by starting from a sequence of functions $g_i^1, \ldots, g_i^k$, and then defining $g_i(y)$to be $h_i^j(y)$ for the first j such that $h_i^j(y) \notin f^{(-1)}(L)$.

2.1.2 Scaling Down the Fiat-Naor Construction and Efficient k-wise Independent Hash Functions

Consider now the issue of scaling down this construction in order to invert only ϵN points.

If we fix parameters r, m, t, ℓ such that $r \cdot m \cdot t = \epsilon N$ and $m \cdot t^2 \leq \ell$, then we have an algorithm that inverts the function at ϵN points and whose time complexity is $t^2 + rt$ and whose space complexity is $\ell + rm + t$. Some calculations show that this gives a time-space trade-off of $TS^3 = (\epsilon N)^3$.

A first improvement comes by considering that if $|f^{(-1)}(L)| \geq \epsilon N$, then just by constructing L we are done. This means that we may assume that the elements not in L have each at most $\epsilon N/\ell$ pre-images, and there are $(1 - \epsilon)N > N/2$ elements not in $f^{(-1)}(L)$, meaning that the collision probability of f restricted to $\{0,1\}^n - f^{(-1)}(L)$ is at most ϵ/ℓ. This is a stronger bound than the "effective collision probability" bound $1/\ell$ in the Fiat-Naor analysis. This means that we can set the parameters so that $rmt = \epsilon N$, $mt^2 \leq \ell/\epsilon$, and have $S = \ell + rm + t$ and $T = t^2 + rt$. This leads to the improved trade-off $TS^3 = \epsilon^4 N^3$, provided $\epsilon N > T > \epsilon^{-2}$.

A second improvement comes by using new constructions of t-wise independent hash functions that can be evaluated in time negligible in t. We present such a construction in Appendix 8. Using such a construction, the running time of the algorithm becomes just rt, rather than $t^2 + rt$. In the original Fiat-Naor construction, the two bounds are of the same order, because optimizing the parameters always leads to $r > t$. In the scaled-down construction we described above, however, $r > t$ is optimal only as long as $T > \epsilon^{-2}$, which is why we added such a constraint above. Hence, we require a family of hash functions with two properties:

- *Small size:* it is sufficient for our purposes that each function be representable with $\Theta(t) + N^{o(1)}$ bits;
- *Efficient evaluation:* given the description of a function in the family and a point in the domain, we would like the evaluation of the function at that point to take time $t^{o(1)} \cdot N^{o(1)}$

We note that most known constructions with small size do not satisfy the efficient computation requirement. The only ones known to us, which satisfy both the properties is the construction by Ostlin and Pagh [OP03] but their construction can differ from being uniform on a set of size t by an inverse polynomial in t. This error is too large for us and hence we come up with a new construction of t-wise independent function family which satisfy both these properties. (More about our construction in Section 2.5 below.) Using these hash functions would lead to the same trade-off $TS^3 = \epsilon^4 N^3$, but for the wider range of parameters $\epsilon N > T > \epsilon^{-1}$.

2.1.3 The Main New Idea

Our main improvement over the techniques of Fiat and Naor comes from the use of a more precise counting of the number of inputs x such that $f(x)$ can be inverted using a given data structure.

We note that if we have the endpoints of a path of length t in our data structure, then we are able to invert f not just at t inputs, but rather at as many inputs as the sum of the indegrees (in G_f) of the vertices of the path.[4] If, for example, the function f is k-regular (meaning that, for every x, $f(x)$ has exactly k pre-images), then a special case of the analysis that we provide shows that we can invert everywhere with trade-off $TS^2 = N^2/k^2$, while the Fiat-Naor analysis would give a trade-off $TS^2 = N^2 \cdot k$. They are the same when $k = \tilde{O}(1)$, but for larger k the analysis of Fiat and Naor provides worse bounds, because the collision probability increases, while our analysis provides better bounds.

For functions that are not regular, providing a good bound on the number of elements that are inverted by the data structure is more challenging.

If the function has collision probability λ (or "effective" collision probability λ after discounting the elements in the high-indegree table), and we construct r data structures, each having m paths of length t, then the average sum of the indegrees of the vertices in the data structure is $m \cdot t \cdot r \cdot \lambda \cdot N$, which is potentially much more than mtr if the collision probability is large. It seems, then, that we could fix parameters m, t, r such that

$$\begin{aligned} & m \cdot t^2 \leq \lambda^{-1} \\ & m \cdot t \cdot r \cdot \lambda N \geq \epsilon N \end{aligned} \tag{1}$$

[4]Said differently, Fiat and Naor count the number of y which are inverted, while one should count the number of x such that $f(x)$ is inverted.

and be able to invert ϵN elements using time rt and space $rm + t$. This would optimize, in the interesting case in which space and time are equal, to having space and time $\max\{\sqrt{\epsilon N}, \epsilon N^{2/3}\}$, which would be great. In particular, it would improve the Fiat-Naor construction even when $\epsilon = 1$. Unfortunately, while $mrt\lambda N$ is the expectation of the sum of the indegrees of the vertices in all the paths of the data structure, it is not the expectation of the number of x such that $f(x)$ is inverted: the problem is that, if the collision probability is very high, there might be elements y with many pre-images that occur in multiple data structures, and which would then be counted multiple times.

We then proceed by considering three cases. If the collision probability is small, that is, less than ϵ^2/S, where S is the amount of space we plan to use, then we find parameters m, t, r such that

$$mt^2 \leq S/\epsilon^2$$
$$mtr \geq \epsilon N$$

that is, we take advantage of the bound on collision probability but we do not attempt to improve the Fiat-Naor count on the number of inverted elements. This allows us to invert an ϵ fraction of elements using time and space at most $\max\{\sqrt{\epsilon N}, \epsilon^{5/4}N^{3/4}\}$.

If the collision probability is more than ϵ^2/S, then we consider how much $mrt\lambda N$ is overcounting the real number of inverted elements. The overcounting is dominated by the elements x such that, for a given choice of r, m, t, $f(x)$ has probability $\Omega(1)$, say, probability $\geq 1/100$, of belonging to one of the data structures. Call such a $y = f(x)$ a *heavy* image to invert.

If the number of pre-images of heavy elements is at least $100\epsilon N$, then we are done, because we expect to be able to invert at least a $1/100$ fraction of heavy elements.

The remaining case, then, is when the collision probability is more than ϵ^2/S, but the total number of preimages of heavy elements is less than $100\epsilon N$. This information, together with the fact that (thanks to the size-S high-indegree table) we are only trying to invert elements with at most $\epsilon N/S$ preimages, allows us to bound the total number of occurrences of heavy elements in the data structure, and to conclude that the total number of pre-images of non-heavy elements is at least $\Omega(mrt\lambda N)$. This means that a choice of m, r, t satisfying (1) leads us to invert an ϵ fraction of inputs, and to do so with time and space at most $\max\{\sqrt{\epsilon N}, \epsilon N^{2/3}\}$.

Applying these ideas to get full time-space trade-offs give us that, if $\epsilon < 1/N^{1/3}$ we can have the optimal trade-off $TS = \epsilon N$; otherwise we achieve the trade-off $TS^3 = \epsilon^5 N^3$.

We remark that for most of the allowed range of the parameters we have $r < t$, and that in the "low-end" range $\epsilon < N^{-1/3}$ for which our result is optimal we have $r = 1$. For this reason there is a notable improvement in using our efficient hash functions instead of the amortized hash function evaluation of Fiat and Naor.

2.2 Pseudorandom Generators

The starting point of our result for pseudorandom generators is the fact [AGHP92] that if two random variables ranging over $\{0,1\}^m$ have constant statistical distance, then there is a linear function (of $O(m)$ circuit complexity) that distinguishes the two random variables with advantage at least $2^{-m/2}$.

Suppose that we are given a length-increasing pseudorandom generator $G : \{0,1\}^{n-1} \to \{0,1\}^n$ and that we want to construct a distinguisher achieving distinguishing probability ϵ.

Our idea is to partition $\{0,1\}^n$ into $\epsilon^2 2^n$ sets each of size ϵ^{-2}, for example based on the value of the first $n - 2\log 1/\epsilon$ bits, and then apply within each block the linear function that provides, within that block, the best distinguishing probability. Overall, this defines a function of circuit complexity $O(\epsilon^2 \cdot n \cdot 2^n)$. Then, intuitively, within each block we achieve distinguishing probability at least ϵ, because each block is a set of size ϵ^{-2}, and the distinguishing probability is at least the square root of the inverse of the block size.

The straightforward implementation of this intuition would be to use, in each block, the linear function that best distinguishes the uniform distribution within the block from the conditional distribution of the output of the generator conditioned on landing in the block. Unfortunately this approach would not work because the overall distinguishing probability is not a convex combination of the conditional distinguishing probabilities.[5]

Instead, in each block, we choose the linear function that most contributes to the overall distinguishing probability. In order to quantify this contribution we need a slight generalization of the result of [AGHP92].

We then present a more efficient distinguisher of circuit complexity $O(\epsilon^2 \cdot 2^n)$ which employs a hash function sampled from a 4-wise independent family, and whose analysis employs a more involved fourth-moment argument, inspired by [ACR97].

2.3 Lower Bounds

Using techniques of Yao [Yao90], Gennaro and Trevisan [GT00], and Wee [Wee05], it is possible to show that, in the generic oracle setting that we consider in this paper, there are permutations for which the amount of advice S and the oracle query complexity T must satisfy

$$S \cdot T \geq \tilde{\Omega}(\epsilon N)$$

for any algorithm that inverts an ϵ fraction of inputs. Such lower bound proofs are based on the idea that an algorithm with better performance could be used to encode every permutation $f : [N] \to [N]$ using strictly less than $\log N!$ bits, which is impossible. Here, we simplify such proofs by using randomized encodings. (Even a randomized encodings cannot represent every permutation using less than $\log N!$, and showing that such an encoding would be possible if the lower bound were wrong is easier by using randomization.) In fact, while previous proofs gave a lower bound on the trade-off only when $T = \tilde{O}(\sqrt{\epsilon N})$, our lower bound works for the full range of parameters.

We then consider the question of the security of pseudorandom generators in the oracle setting. By using the known results for permutations and applying efficient hard-core predicates it could be possible to show the existence of generators for which $S \cdot T \geq \epsilon^7 N$. By instead applying the ideas of randomized encodings to a pair f, p where f is a random permutation (modeling a one-way permutation) and p is a random predicate (modeling a "hard-core predicate" for p), we prove the existence of length-increasing generators such that for every distinguisher that makes T oracle queries to the generator, and which has advice S and distinguishing probability ϵ, we have

$$S \cdot T \geq \tilde{\Omega}(\epsilon^2 N)$$

where N is the number of seeds. There is still a gap, even for generators of the form $x \to f(x)p(x)$ where f is a permutation, between our lower bound and known constructions of distinguishers.

[5]This is a subtle issue related to the fact that the condition of landing in a given block might have different probabilities in the uniform distribution versus the output of the generator. If so, then the respective conditional probabilities are normalized differently, and the use of a distinguisher for the conditional distributions in a block does not necessarily contribute to the task of distinguishing the original distributions.

In particular, the best known algorithm is one of the following (depending on ϵ, S, T) : Use the algorithm for inverting functions which can be at best $S \cdot T \leq \tilde{\Omega}(\epsilon N)$ or use the circuit which we described in the previous subsection. That in particular uses $S = \tilde{\Omega}(\epsilon^2 N)$ and $T = \tilde{\Omega}(1)$.

Finally, we look at the *common random string* model, in which all parties share a common random string k, which they can use to select a permutation $f_k(\cdot)$ from a family of permutations, or a generator $G_k(\cdot)$ from a family of generators. In such a setting, the trivial brute force attack that achieve inverting (and distinguishing) probability $\epsilon = T/N$ with no advice remains possible. Alternatively, one can think of a family of permutations as a single permutation $(k, x) \rightarrow (k, f_k(x))$. We show that, for families of permutations, either the trivial uniform algorithm or Hellman's construction applied to the mapping $(k, x) \rightarrow (k, f_k(x))$ are best possible, depending on whether the available advice is shorter or longer than the number of keys.

For generators in the common random string model we show that either $T \geq \tilde{\Omega}(\epsilon^2 N)$ or $ST \geq \tilde{\Omega}(\epsilon^2 KN)$, where K is the number of keys.

2.4 Uniformity and Model of Computation

Positive Results

In the time/space trade-offs of Hellman, of Fiat and Naor, and of this paper, an algorithm uses "time T" and "space S" if it runs in time at most T (in a RAM model), uses at most S bits of space, and works correctly upon receiving S bits of advice, in the form of an S-bit data structure that dominates the space requirement of the algorithm.

The "advice," in turn, can be computed in uniform time $\tilde{O}(N)$. In the work of Hellman and of Fiat and Naor, one cannot hope, in general, to have processing time significantly smaller than N in order to generate the data structure used by the algorithm. Otherwise, one would have a uniform algorithm that inverts an arbitrary one-way permutation (or function) in time noticeable smaller than N, which is impossible relative to a random permutation (or function) oracle.

In our paper, the data structure we use is also easily pre-computable in time $\tilde{O}(N)$. Pre-processing time significantly smaller than ϵN should not be expected, because then we would have a uniform algorithm to invert a random function on an ϵ fraction of inputs in time significantly smaller than ϵN. With some care, our data structure can indeed be pre-computed using optimal uniform time $\tilde{O}(\epsilon N)$. We, however, do not describe it here for the sake of simplicity.

Negative Results

When we show that a particular combination of space S and time T is not achievable, our result rules out non-uniform algorithms that make at most T oracle queries to the function (or generator) oracle, and which receive at most S bits of advice. The actual space used by the algorithm, as well as the complexity of the computations performed between oracle queries, can be unbounded. Likewise, the non-uniform advice can have arbitrary complexity.

2.5 Efficient k-wise Independent Hash Functions

It is important for our results to have families of k-wise independent hash functions that can be represented using an $\tilde{O}(k)$-bit seed and can be evaluated in $k^{o(1)}$ time. Ostlin and Pagh [OP03] present very efficient construction of almost k-wise independent hash functions that can be evaluated in constant time on a RAM. In their construction, however, a k-tuple of evaluations can have a distance from the uniform distribution which is inverse polynomial in k, which is too large for our purposes.

We construct a family of efficient k-wise independent hash functions based on the unique-neighbor expander graphs of by Capalbo *et al.* [CRVW02]. We note that, in order to construct a family $\{h_r\}$ of k-wise independent functions $h_r[N] \to \{0,1\}$ indexed by a seed $r \in \{0,1\}^t$, it is enough to construct N vectors $a_1, \ldots, a_N$ in $\{0,1\}^t$ with the property that any k of them are linearly independent, and then define $h_r(x) := \langle r, a_x \rangle$, where $\langle \cdot, \cdot \rangle$ denotes inner product mod 2. If the vectors are sparse, and one has an efficient way of generating the list of non-zero coordinates of the vector a_x given x, then the computation of $h_r(\cdot)$ can take time $r^{o(1)}$ time. In our construction, we choose the vectors to be rows of the adjacency matrix of the unique-neighbor bipartite expanders of Capalbo *et al.*

3 The Fiat-Naor construction and our modification

We first describe the scheme for inverting one-way functions described by Fiat and Naor [FN99]. While our aim is to invert only a fraction of the inputs, Fiat and Naor are interested in inverting f on all the inputs. Like Helllman, they also consider the composition $h = g \circ f$ for a suitably chosen g (which was a random function for Hellman's scheme) and store *walks* (sequences of iterates) according to h. We define the notion formally below.

Definition 3.1 *Let f, g be two functions and let $h = g \circ f$. By a walk of length t starting at x, we mean the sequence of points $(x, h(x), \ldots, h^t(x))$. We say that for a point z, the walk inverts $f(z)$ if there exists $i \le t-1$ such that $f(h^i(x)) = f(z)$ i.e. one of the points in the walk is an inverse of $f(z)$.*

The sequence above can be viewed as a walk if one considers the graph with all $x \in [N]$ as vertices and directed edges from x to $h(x)$. The outdegree of every vertex in the graph is 1 and the indegree is exactly the number of pre-images of the element corresponding to the vertex. Fiat and Naor store the start and end points $(\{x, h^t(x)\})$ of m walks according to $h = g \circ f$ for a sufficiently random function g. Given an element y, they construct the sequence $g(y), h(g(y)), \ldots, h^{t-1}(g(y)$. If the sequence hits the endpoint for any of the stored walks, they searh the the stored walk from the start looking for the inverse of y. Composing f with a random function g helps to ensure that the walks cover a large portion of the graph and hence invert many elements in the range of f.

However, naively implementing this scheme does not work if the distribution of indegrees in the graph is highly skewed. Note that for any g, the distribution of pre-images for the function $h = g \circ f$ is at least as skewed as the distribution for f. Having a very skewed h can create the following problem: suppose that f maps half of the points in the domain to one point and is a permutation among the others. This means that irrespective of the randomness of g, the walk starting from any point x_i is going to cycle within a constant number of steps with very high probability. Hence, one can only invert $\Theta(m)$ points from the image of permutation using m walks. Thus to invert all the points, one needs to make $m = \Theta(N)$.

The strategy in [FN99] to take care of this problem is to construct a list L of size ℓ ($\ell = \tilde{O}(N^{3/4})$ in [FN99]) in which they store ℓ randomly chosen points (and their inverses) with probability proportional to the number of pre-images. Also, the function $g : [N] \to [N]$ is chosen to be a (pseudo)random function conditioned on having its images outside the list. The walks are then used to only invert the elements which are outside the list (and are likely to have only few pre-images) while the others can be inverted by searching the stored list L.

However, in the Fiat-Naor construction, the parameters m and t cannot be chosen to be too large. If the length of the walks t becomes too large then the walks my cycle and their analysis does not work for this case. Also, when the the number of walks is too large, starting the walk from a given point y, one may reach the endpoint of many stored walks. It is then not clear which of the walks to search to find an inverse. To boost the probability of inversion, they repeat the entire construction r times according to r ($r \approx O(N^{1/3})$ for their construction) randomly chosen functions $g_1, \ldots, g_r$.

One subtlety missing in the above description is that the functions $g_1, \ldots, g_r$ are actually chosen according to a family of $O(t)$-wise independent functions. Given an element y to invert, one needs to compute the sequences $(g_i(y), h_i(g_i(y)), \ldots, h_i^{t-1}(g_i(y)))$ for all $i \in [r]$. It is not clear how to compute each function in $\tilde{O}(1)$ time on a given input to achieve a total running time of $\tilde{O}(r \cdot t)$. However, their functions are chosen from a special family so that the r calls to $g_1, \ldots, g_r$ can be computed in total $\tilde{O}(r+t)$ time. This results in a total running time of $\tilde{O}(t^2 + tr)$. As $t = O(r)$ in the setting in [FN99] (but not ours), they achieve a total running time of $\tilde{O}(tr)$.

Our modification

As mentioned before, we only seek to invert f at an ϵ fraction of the inputs rather than all the inputs (as was the case with Fiat and Naor [FN99]). In case of inverting all the inputs, the major bottleneck is not the time and space for computing walks (which is about $N^{2/3}$) but the space for storing list consisting of the high in-degree elements to prevent walks from cycling. This requires space $N^{3/4}$ in the worst case. Since, we want to invert only ϵ fraction of elements, one can hope to accomplish this with a much smaller table (list) space.

The basic structure of our construction is similar to the one by Fiat and Naor. We construct a data structure similar to theirs (see Figure 2) with parameters ℓ, m, t and r as described above. However our analysis differs from theirs in accounting for the number of inputs which can be inverted by presence of a single entry in the data structure. In particular, if an element y is present in a walk, then helps invert f on $\left|f^{-1}(y)\right|$ many inputs. This fact is exploited in our analysis.

Another point of difference in the family of functions from which $g_1, \ldots g_r$ are selected. As mentioned before, Fiat and Naor [FN99] use $O(t)$-wise independent functions. However, off-the-shelf constructions of t-wise independent functions take $\Theta(t)$ time for evaluation at a single point and time for computing each walk would then be $O(t^2)$ instead of $O(t)$. In the Fiat-Naor this problem was solved by choosing $g_1, \ldots, g_r$ from a specific family so that one evaluation of each of the r functions could be performed in total $\tilde{O}(t+r)$ time. In our case, when ϵ is small, then the optimum parameters of the data structure are achieved with $r = 1$ and hence amortizing over different functions is not an option.

We get rid of this problem by constructing a family of k-wise independent functions (we shall need $k = 2t \cdot (\log N)^2$) such that evaluation time for any of the functions from the family is $2^{O((\log\log N)^3)} = N^{o(1)}$. This is good for us as this is $k^{o(1)}$ because we always have $k = N^{\Theta(1)}$ (As previously mentioned, our bounds are anyway specified hiding factors of $N^{o(1)}$). The construction is made possible due to a remarkable construction of lossless expanders by Capalbo *et al.* [CRVW02]. The construction of the functions is described in Appendix 8. We only need the following fact from that section.

Theorem 3.2 *For any $N, N' \in \mathbb{N}$ and $k \leq N^{0.99}$, there exists an explicit construction of a family*

Parameters

ℓ	:=	Size of list L
t	:=	Length of each walk
r	:=	Number of independent functions $g \in \mathcal{F}$ used
m	:=	Number of walks according to each function g_i

Construction of Data Structure

1. Consider the ℓ elements in the range of f with highest value of $I(y)$. For each such element y, store an entry (y, x) in the list for some $x \in f^{-1}(y)$.
2. Choose functions $g_1, \ldots, g_r \in \mathcal{F}$ pairwise independently at random. For each function g_i, define the partial function $g_i^* : [N] \to [N]$ as

$$g_i^*(x) \;=\; \begin{cases} g_i(x,u) & \text{if } u \text{ is the least index such that } f(g_i(x,u)) \notin L \\ \text{undefined} & \text{if } \forall u \in [(\log N)^2].\ f(g_i(x,u)) \in L \end{cases}$$

   For each i, define the partial function $h_i = g_i^* \circ f$.
3. For each $i \in [r]$ and $j \in [m]$, construct a walk W_{ij} of length t; by starting at a random point x_{ij} and computing the sequence $x_{ij}, h_i(x_{ij}), \ldots, h_i^t(x_{ij})$. Discard the walk if
   - for some $t_1 \le t$, $h_i^{t_1}(x_{ij})$ is undefined.
   - the walk cycles i.e. for $t_1, t_2 \le t$, $h_i^{t_1}(x_{ij}) = h_i^{t_2}(x_{ij})$.

   For walks W_{ij} that are not discarded, store the pairs $(x_{ij}, h_i^t(x_{ij}))$.

Figure 2: Description of data structure for inverting f

$\mathcal{F} : [N] \times [N'] \to [N]$ of k-wise independent functions such that the randomness required to sample $g \in \mathcal{F}$ is $k2^{\Theta((\log\log N + \log\log N')^3)}$. Further, for any such $g \in \mathcal{F}$, on any input x, $g(x)$ can be computed in time $2^{\Theta((\log\log N + \log\log N')^3)}$.

We give a description of the data structure we construct in Figure 2. The functions $g_1, \ldots, g_r : [N] \times [n^2] \to [N]$ are chosen from a family $\mathcal{F}$ of $k = 2t \cdot (\log N)^2$-wise independent functions, as described in Theorem 3.2.

We use $I(y)$ to denote the number of pre-images of an element y according to the function f. The list L contains the ℓ entries of the form (x, y) for the ℓ elements in the range of f with the highest values of $I(y)$, and an arbitrary $x \in f^{-1}(y)$. We say that $y \in L$ if $(x, y) \in L$ for some x. One parameter which will be useful in our analysis is the *collision probability* of the ditribution of pre-images for the elements y not in L, which we denote by λ_ℓ. Let $\tilde{N} = N - \sum_{y \in L} I(y)$. Then,

$$\lambda_\ell \;:=\; \sum_{y \notin L} \left(\frac{I(y)}{\tilde{N}}\right)^2 \tag{2}$$

The procedure for inversion

We now describe the algorithm used for inverting f on a given y using the data structure. Given a y, we treat y as $f(x)$ for an unknown x and compute r walks of length t starting at x (note that we only need to know $f(x)$ to compute the walk) according to the functions $h_1, \ldots, h_r$. If we hit

```
Invert(y)
  1. If $(x, y) \in L$ for some $L$, return $x$.
  2. For each $i \in [r]$
     (a) Construct the sequence $(g_i^*(y), h_i(g_i^*(y)), \ldots, h_i^{t-1}(g_i^*(y)))$.
     (b) If there are indices $j_0 \in [m]$ and $t_0 \le t-1$ such that $h_i^{t_0}(g_i^*(y)) = h_i^t(x_{ij_0})$, then compute
         $h_i^{t-t_0-1}(x_{ij_0})$. In case there are multiple choices for $j_0$, pick the smallest one.
     (c) If $f(h_i^{t-t_0-1}(x_{ij_0})) = y$, output $h_i^{t-t_0-1}(x_{ij_0})$ else output fail.
```

Figure 3: Procedure for inverting a given element y

the endpoint of a stored walk according to any of the r functions, we compute the walk from the stored starting point and check if it contains the inverse of y.

Note that we check only one of the walks $W_{i1}, \ldots, W_{im}$ according to a function h_i. In particular, if the sequence $(g_i^*(y), h_i(g_i^*(y)), \ldots, h_i^{t-1}(g_i^*(y)))$ contains the endpoints of two of the stored walks, we pick one arbitrarily and search that. This requires that our data structure avoids *false hits* for y, which we define formally below. We shall be concerned with the probability of false hits in our analysis of the data structure.

Definition 3.3 *We say that a walk $W = (x, h(x), \ldots, h^t(x))$ for $h = g \circ f$ produces a false hit for y if the sequence $(g(y), h(g(y)), \ldots, h^{t-1}(g(y)))$ does contain the endpoint $h^t(x)$ of W, but the walk does not invert y.*

4 Analysis of the Data Structure

In this section, we establish the constraints on the parameters of the data structure such that if these constraints are satisfied, then the number of elements $x \in [N]$ such that $f(x)$ is contained in one of the walks is at least ϵN. In particular, we prove the following theorem (recall that λ_ℓ is the "effective collision probability" defined in (2)):

Theorem 4.1 *For given $\epsilon > 0$ and $f : [N] \to [N]$, let the parameters m, t, ℓ in the data structure be such that $mt^2\lambda_\ell < 1/8$ and $\epsilon t/\ell < 1/4$. In case, the high in-degree list does not invert an ϵ fraction trivially, then, there exists an $r = O(\epsilon N/mt)$ such that*

$$\mathop{\mathbb{P}}_{x \in [N]} \left[\mathsf{Invert}(f(x)) \in f^{-1}(f(x))\right] \;\ge\; \epsilon$$

Moreover, if $\ell \cdot \lambda_\ell \ge 100\epsilon^2$, then r can be taken to be $O(\epsilon/mt\lambda_\ell)$. In the above expression, the probability includes the probability over the randomness of the data structure. Further $\lambda_\ell \le 2\epsilon/\ell$ subject to the list itself not inverting an ϵ fraction of the elements.

To prove the theorem, we first prove various claims regarding the probability of a single input being inverted by the data structure. Throughout the analysis, we will assume that $\epsilon \le c$ where c is an arbitrarily small constant (The value $c = \frac{1}{1000}$ suffices). The case $\epsilon \ge c$ can be handled by simply using the scheme in [FN99] which inverts all the elements. For large ϵ, the parameters we obtain are the same as in [FN99] up to a multiplicative constant.

4.1 Inversion by the list

We are interested in the fraction of elements x such that $f(x)$ is inverted by the data structure. If the list itself does not suffice to invert $f(x)$ on ϵN inputs, it does ensure that elements outside the list have only a small number of pre-images. We formalize this in the claim below

Claim 4.2 *If the list L does not invert f on ϵ fraction of the inputs i.e. $\mathbb{P}_{x\in[N]}[f(x) \in L] < \epsilon$, then*

1. $\sum_{y\notin L} I(y) \;\geq\; (1-\epsilon)N$
2. *For all elements $y \notin L$, $I(y) \leq \epsilon N/\ell$*
3. $1/N \leq \lambda_\ell \leq 2\epsilon/\ell$

Proof: To prove the claim we simply need to observe that each element $y \in L$ contributes exactly $I(y)/N$ to $\mathbb{P}_{x\in[N]}[f(x) \in L]$. Since the probability is less than ϵ by assumption, we get that $\sum_{y\in L} I(y) < \epsilon N$. Also, since f is a total function, $\sum_{y\in[N]} I(y) = N$ which proves the first part.

To prove the second part, we note that since the list contains the ℓ elements with the highest values of $I(y)$, for any element $y \notin L$, $I(y) \;\leq\; \mathbb{E}_{z\in L}[I(z)] \;<\; \epsilon N/\ell$.

Recall that $\tilde{N} = N - \sum_{y\in L} I(y)$ denotes set $\{x \in [N] : f(x) \notin T\}$. The following gives the desired upper bound on λ_ℓ.

$$\lambda_\ell = \sum_{y\notin\ell}\left(\frac{I(y)}{\tilde{N}}\right)^2 \leq \left(\sum_{y\notin\ell}\frac{I(y)}{\tilde{N}}\right)\left(\max_{y\notin\ell}\frac{I(y)}{\tilde{N}}\right) \leq \frac{\epsilon N}{\ell\tilde{N}} \leq \frac{2\epsilon}{\ell}$$

The last inequality uses that $\tilde{N} \geq N/2$. The lower bound on λ_ℓ follows simply from the fact that λ_ℓ is the collision probability of a function over a domain of size $\tilde{N} \leq N$. As collision probability of any function over a domain of size $\tilde{N}$ is at least $1/\tilde{N}$, the lower bound follows. ■

4.2 Inversion by walks

We first restrict ourselves to the case $r = 1$ and consider m walks $W_1, \ldots, W_m$ according to a single function $g \in \mathcal{F}$ and the corresponding partial function $h = g^* \circ f$.

We shall be interested in computing the expected (over the randomness in the construction of the data structure) number of elements x such that $f(x)$ is inverted by the data structure. To this end, we first compute the probability that a single walk $W = (x, h(x), \ldots h^t(x))$ inverts a given element $y \notin L$(and is well defined). Recall that we say that the walk inverts y if there is a point $h^j(x)$ in the walk for $j \leq t-1$ such that $f(h^j(x)) = y$. All probabilities below shall be over the construction of the data structure, unless specified otherwise.

Claim 4.3 *Let $W = (x, h(x), \ldots, h^t(x))$ be a walk constructed using randomly chosen x and h. For a given $y \notin L$*

$$\mathbb{P}[W \text{ is not discarded and inverts } y] \;\geq\; \frac{tI(y)}{2\tilde{N}}\left(1 - \frac{tI(y)}{\tilde{N}} - t^2\lambda_\ell\right)$$

Proof: Recall that a walk is discarded if it is undefined i.e. for some $j \in [t]$, $h^j(x)$ is undefined. For a given input z, $h(z)$ is undefined if $f(g(x,u)) \in L$ for all $u \in [(\log N)^2]$. Using the fact that $g(x,u)$ is uniformly distributed

$$\mathbb{P}[f(g(x,u)) \in L] \;=\; \mathop{\mathbb{P}}_{x \in [N]}[f(x) \in L] \;<\; \epsilon$$

Also, since g is $2t \cdot (\log N)^2$-wise independent, $\mathbb{P}[g(z,u) \in T \text{ for all } u] < \epsilon^{(\log N)^2}$. By a union bound, $\mathbb{P}[W \text{ is undefined}] \leq t \cdot \epsilon^{(\log N)^2}$.

We now compute the probability that the walk inverts y and does not cycle, given that it is defined at each point. We can split this event into events $f(h^j(x)) = y$ for $0 \leq j \leq t-1$, which are *disjoint* since we also include the condition that the walk does not cycle. We then get

$$\begin{aligned}
&\mathbb{P}\left[W \text{ does not cycle and inverts } y \mid W \text{ is defined}\right] \\
&= \sum_{j=0}^{t-1} \mathbb{P}\left[f(h^j(x)) = y \;\&\; W \text{ does not cycle} \mid W \text{ is defined}\right] \\
&= \sum_{j=0}^{t-1} \mathbb{P}\left[f(h^j(x)) = y \mid W \text{ is defined}\right] \cdot \mathbb{P}\left[W \text{ does not cycle} \mid f(h^j(x)) = y \;\&\; W \text{ is defined}\right] \\
&= \sum_{j=0}^{t-1} \frac{I(y)}{\tilde{N}} \cdot \mathbb{P}\left[W \text{ does not cycle} \mid f(h^j(x)) = y \;\&\; W \text{ is defined}\right]
\end{aligned}$$

since $f(h^j(x))$ is uniformly distributed outside the list L, when conditioned on W being defined (using uniform distribution of $h^j(x)$).

We now consider the probalilty of cycling. Let $j_1 < j_2$ be the smallest indices such that $f(h^{j_1}(x)) = f(h^{j_2}(x))$. If either $\min\{j_1, j_2\} \leq j$, then there exists an index $j' \leq 2t, j' \neq j$ such that $f(h^{j'}(x)) = f(h^j(x)) = y$. We can bound this probability using the estimate above and a union bound.

$$\mathbb{P}\left[\exists j' \leq 2t, j' > j \;s.t.\; f(h^{j'}(x)) = y \mid f(h^j(x)) = y \;\&\; W \text{ is defined}\right] \;\leq\; \frac{2tI(y)}{\tilde{N}}$$

We now need to bound the probability of cycling when $f(h^{j_1}(x)) = f(h^{j_2}(x))$ for $j < j_1 < j_2$. But for j_1 and j_2 both greater than j, $f(h^{j_1}(x))$ and $f(h^{j_2}(x))$ are both independently distributed images of f outside L, even when we condition on $f(h^j(x)) = y$. Hence the probability of their being equal is exactly λ_ℓ, which was defined to be the collision probability of elements outside L. This gives

$$\mathbb{P}\left[\exists j_1, j_2 > j \;s.t.\; f(h^{j_1}(x)) = f(h^{j_1}(x)) \mid f(h^j(x)) \;\&\; W \text{ is defined}\right] \;\leq\; t^2 \cdot \lambda_\ell$$

Thus we get a bound of $2tI(y)/\tilde{N} + t^2\lambda_\ell$ on the probability of cycling. Using the previous bounds, this gives

$$\mathbb{P}\left[W \text{ is not discarded and inverts } y\right] \;\geq\; (1 - t \cdot \epsilon^{(\log N)^2}) \cdot \frac{tI(y)}{\tilde{N}} \cdot \left(1 - \frac{2tI(y)}{\tilde{N}} - t^2 \cdot \lambda_\ell\right)$$

which proves the claim since $\epsilon < 1/2$ and $t < N$ suffices to conclude $t \cdot \epsilon^{(\log N)^2} < 1/2$. ■

We will be interested in computing the probability that at least one of the walks inverts y. Two walks can interfere with each other in two ways in the inversion process. The first interference, which is in terms of the analysis, is that we need to account for the probability of two walks both inverting y to get a lower bound on the probability of at least one walk inverting y.

The second interference is in terms of the procedure of inversion. Let W_1 and W_2 be two walks generated according to the same function h and different starting points x_1 and x_2. Then, if $f(h^j(x_1)) = y$ and $f(h^{j_1}(x_1)) = f(h^{j_2}(x_2))$ for some $j_1 > j$, then although the walk W_2 does not invert y, we may see both the points $f(h^t(x_1))$ and $f(h^t(x_2))$ if we start in the inversion procedure from $g^*(y)$ and take a t-step walk. This is the event we call a *false hit*, which affects the running time of the inversion procedure.

In the claim below, we derive upper bounds on the probabilities of both kinds of interference mentioned above.

Claim 4.4 *Let $W_1 = (x_1, h(x_1), \ldots, h^t(x_1))$ and $W_2 = (x_2, h(x_2), \ldots, h^t(x_2))$ be two walks generated according to randomly chosen h, x_1 and x_2. Then for any $y \notin L$,*

1. $\mathbb{P}\left[W_1, W_2 \text{ are not discarded and both invert } y\right] \;\leq\; \left(\frac{tI(y)}{\tilde{N}}\right)^2$
2. $\mathbb{P}\left[W_1 \text{ inverts } y \text{ \& } W_2 \text{ generates a false hit}\right] \;\leq\; \frac{t^3 I(y)\lambda_\ell}{\tilde{N}}$

Proof: To prove the first part of the claim, we simply note that the $2t$-positions in the two walks are independent. Also, conditioned on the walks being defined, the points in the walks are uniformly distributed over the pre-images of elements not in L. This gives

$$\begin{aligned}
&\mathbb{P}\left[W_1, W_2 \text{ are not discarded and both invert } y\right] \\
&\leq \sum_{j_1, j_2 < t} \mathbb{P}\left[f(h^{j_1}(x_1)) = f(h^{j_2}(x_2)) = y \mid W_1, W_2 \text{ are defined}\right] \;=\; t^2 \cdot \left(\frac{I(y)}{\tilde{N}}\right)^2
\end{aligned}$$

For W_1 to invert y, for some index j we must have $f(h^j(x_1)) = y$. Also, to have a false hit, we must have two indices j_1, j_2 such that $j_1 > j$ and $f(h^{j_1}(x_1)) = f(h^{j_2}(x_2))$ (which will not equal y if the walks do not cycle). Also, both the events are independent for fixed j, j_1, j_2 and hence

$$\begin{aligned}
&\mathbb{P}\left[W_1 \text{ inverts } y \text{ \& } W_2 \text{ generates a false hit}\right] \\
&\leq \sum_{j, j_1, j_2} \mathbb{P}\left[\left(f(h^{j_1}(x_1)) = f(h^{j_2}(x_2))\right) \wedge \left(f(h^j(x_1)) = y\right) \wedge (W_1 \text{ doesn't cycle}) \,|W_1, W_2 \text{ are defined}\right] \\
&\leq \sum_{j, j_1, j_2} \mathbb{P}\left[\left(f(h^{j_1}(x_1)) = f(h^{j_2}(x_2))\right) \wedge \left(f(h^j(x_1)) = y\right) \,|W_1, W_2 \text{ are defined}\right] \\
&= \sum_{j, j_1, j_2} \lambda_\ell \cdot \frac{I(y)}{\tilde{N}} \;=\; \frac{t^3 I(y)\lambda_\ell}{\tilde{N}}
\end{aligned}$$

■

We are now ready to prove a lower bound on the probabilty at least one of m walks constructed according to a single function g inverts a given input y. We say that y is *inverted without false hits* by the walks $W_1, \ldots, W_m$, if some W_i inverts y and none of the other walks produce a false hit.

Claim 4.5 *Let $W_1, \ldots, W_m$ be m walks of length t constructed according a function h chosen as before and independent random starting points $x_1, \ldots, x_m$. If $mt^2\lambda_\ell \leq 1/8$ and $\epsilon t/\ell \leq 1/4$, then for any $y \notin L$*

$$\mathbb{P}\left[y \text{ is inverted without false hits by } W_1, \ldots, W_m\right] \;\geq\; \min\left(\frac{1}{32}, \frac{mtI(y)}{4\tilde{N}}\right)$$

Proof: We will bound this probability by inclusion-exclusion. To express the required probabilities, we first define the events $E_i(y)$ and $F_i(y)$ as below

$$\begin{aligned} E_i(y) &:= [W_i \text{ inverts } y] \\ F_i(y) &:= [W_i \text{ produces a false hit for } y] \end{aligned}$$

It is clear that for any i, the events E_i and F_i are disjoint. We first express the required probability in terms of the above events.

$$\mathbb{P}\left[y \text{ is inverted without false hits by } W_1, \ldots, W_m\right] \;=\; \mathbb{P}\left[\left(\bigvee_{i=1}^{m} E_i(y)\right) \wedge \overline{\left(\bigvee_{i=1}^{m} F_i(y)\right)}\right]$$

For convenience of analysis, we will consider an $m' \leq m$ and consider inversion only by the first m' walks (however, false hits are still ruled out for all the walks). In particular, for any $m' \leq m$

$$\mathbb{P}\left[\left(\bigvee_{i=1}^{m} E_i(y)\right) \wedge \overline{\left(\bigvee_{i=1}^{m} F_i(y)\right)}\right] \;\geq\; \mathbb{P}\left[\left(\bigvee_{i=1}^{m'} E_i(y)\right) \wedge \overline{\left(\bigvee_{i=1}^{m} F_i(y)\right)}\right]$$

We can now use inclusion-exclusion to bound the expression on the right.

$$\begin{aligned} \mathbb{P}\left[\left(\bigvee_{i=1}^{m'} E_i(y)\right) \wedge \overline{\left(\bigvee_{i=1}^{m} F_i(y)\right)}\right] &\geq \sum_{i=1}^{m'} \mathbb{P}\left[E_i(y) \wedge \overline{\left(\bigvee_{i=1}^{m} F_i(y)\right)}\right] - \sum_{i<j\leq m'} \mathbb{P}\left[E_i(y) \wedge E_j(y)\right] \\ &\geq \sum_{i=1}^{m'} \left(\mathbb{P}\left[E_i(y)\right] - \sum_{j=1, j\neq i}^{m} \mathbb{P}\left[E_i(y) \wedge F_j(y)\right]\right) \\ &\quad - \sum_{i<j\leq m'} \mathbb{P}\left[E_i(y) \wedge E_j(y)\right] \\ &\geq m' \cdot \frac{tI(y)}{2\tilde{N}} \left(1 - \frac{tI(y)}{\tilde{N}} - t^2\lambda_\ell\right) - m'(m-1) \cdot \frac{t^3 I(y)\lambda_\ell}{\tilde{N}} \\ &\quad - \binom{m'}{2} \cdot \left(\frac{tI(y)}{\tilde{N}}\right)^2 \\ &\geq \frac{m'tI(y)}{2\tilde{N}} \left(1 - \frac{m'tI(y)}{\tilde{N}} - 2mt^2\lambda_\ell\right) \end{aligned}$$

In the above computation, we used the lower bound on $\mathbb{P}[E_i(y)]$ from Claim 4.3 and the upper bounds on $\mathbb{P}[E_i(y) \wedge E_j(y)]$ and $\mathbb{P}[E_i(y) \wedge F_j(y)]$ from Claim 4.4. If $mtI(y)/\tilde{N} \leq 1/4$, then we can simply take $m' = m$ and use $mt^2\lambda_\ell \leq 1/8$ to conclude that the required probability is at least $mtI(y)/4\tilde{N}$.

If this is not the case, then choose an $m' \leq m$ such that $1/4 < m'tI(y)/\tilde{N} < 1/2$. The upper bound of $1/2$ is possible to achieve, as from Claim 4.2 and the assumption $\epsilon t/\ell < 1/4$, we get that

$$\frac{tI(y)}{\tilde{N}} \;\leq\; \frac{t}{\tilde{N}} \cdot \frac{\epsilon N}{\ell} \;<\; \frac{1}{2}$$

Using this m' gives that the required probability is at least $1/32$. ■

We now consider r sets of m walks, each set $\mathcal{W}_i = \{W_{i1}, \ldots, W_{im}\}$ constructed according to pairwise independently and randomly chosen function $g_i \in \mathcal{F}$. We are interested in the probability that at least one set inverts a given y without false hits. Note that we only want to rule out false hits *within* each set (the probability for which we have already computed). We do not need to rule out the event that W_{ij} inverts y and $W_{i'j'}$ produces a false hit for some $i \neq i'$, since if there are no false hits within $\mathcal{W}_i$, our algorithm will find an inverse. The following claim is easy to prove using reasoning as before.

Claim 4.6 *Let $\mathcal{W}_1, \ldots, \mathcal{W}_r$ be r sets of m walks each, as above. Also, let $mt^2\lambda_\ell \leq 1/8$ and $\epsilon t/\ell < 1/4$. Then, for any $y \notin L$*

$$\mathbb{P}[y \text{ is inverted without false hits by one of the } r \text{ sets}] \;\geq\; \min\left\{\frac{1}{32}, \frac{rmtI(y)}{8\tilde{N}}\right\}$$

Proof: Let p_y be the probability that y is inverted by a single set without false hits. Since the r sets are constructed according to pairwise independently chosen functions, for any $r' \leq r$

$$\mathbb{P}[y \text{ is inverted without false hits by one of the } r \text{ sets}] \;\geq\; r' \cdot p_y - \binom{r'}{2} \cdot p_y^2$$

Since the conditions for Claim 4.5 are satisfied, we have $p_y \geq \min\left\{1/32, mtI(y)/4\tilde{N}\right\}$. We choose $r' = 1$ if $mtI(y)/4\tilde{N} \geq 1/32$ and $r' = r$ if $rmtI(y)/8\tilde{N} < 1/2$. If neither of these is the case, then choosing an r' such that $\frac{1}{4} \leq \frac{r'mtI(y)}{8\tilde{N}} < \frac{1}{2}$ proves the claim. ■

We can now complete the proof of Theorem 4.1.

Proof (of Theorem 4.1): For an element y, let $p(y)$ denote the the probability over the construction of the data structure that $\mathsf{Invert}(y) \in f^{-1}(y)$. Since every y for which the inverse can be computed by the data structure contributes $I(y)/N$ to the fraction of inputs x for which $f^{-1}(f(x))$ can be computed, we get that

$$\mathbb{E}\left[\mathop{\mathbb{P}}_{x \in [N]}\left[\mathsf{Invert}(f(x)) \;\in\; f^{-1}(f(x))\right]\right] \;=\; \sum_y p(y) \cdot \frac{I(y)}{N}$$

where the expectation is taken over the construction of the data structure. For a fixed r, consider the set

$$Q_r \;:=\; \left\{y \mid y \notin L \text{ and } \frac{rmtI(y)}{8\tilde{N}} > \frac{1}{32}\right\}$$

Then it follows from Claim 4.5 and the definition of the list L that

$$\begin{aligned}\sum_y p(y)\cdot\frac{I(y)}{N} &= \sum_{y\in L}\frac{I(y)}{N} + \sum_{y\in Q_r}\frac{I(y)}{32N} + \sum_{y\in\overline{Q_r\cup L}}\frac{rmtI(y)}{8\tilde{N}}\cdot\frac{I(y)}{N}\\ &\geq \sum_{y\in Q_r}\frac{I(y)}{32N} + \frac{rmt}{10}\cdot\sum_{y\in\overline{Q_r\cup L}}\left(\frac{I(y)}{\tilde{N}}\right)^2\end{aligned}$$

where we used Claim 4.2 to conclude that $\tilde{N}\geq 4N/5$. Let λ_1 denote the quantity $\sum_{y\in\overline{Q_r\cup L}}\left(\frac{I(y)}{\tilde{N}}\right)^2$. For any r, either $\sum_{y\in Q_r}\frac{I(y)}{32N}\geq\epsilon$, or using Claim 4.2 we can get that

$$\lambda_1 = \sum_{y\in\overline{Q_r\cup L}}\left(\frac{I(y)}{\tilde{N}}\right)^2 \geq \frac{1}{\left|\overline{Q_r\cup L}\right|}\cdot\left(\sum_{y\in\overline{Q_r\cup L}}\frac{I(y)}{\tilde{N}}\right)^2 \geq \frac{1}{\left|\overline{Q_r\cup L}\right|}\cdot\frac{(1-33\epsilon)^2N^2}{(1-\epsilon)^2N^2} \geq \frac{1}{2N}$$

Hence, if we choose $r=20\epsilon N/mt$, then either $\sum_{y\in Q_r}\frac{I(y)}{32N}\geq\epsilon$ or $rmt\lambda_1/10\geq\epsilon$. In either case, the data structure (in expectation) inverts f on ϵ fraction of inputs.

If we know that $\ell\cdot\lambda_\ell\geq 100\epsilon^2$, then we can obtain a better bound on λ_1 and hence on r. We observe (using Claim 4.2) that

$$\lambda_2 := \lambda_\ell-\lambda_1 = \sum_{y\in Q_r}\left(\frac{I(y)}{\tilde{N}}\right)^2 \leq \frac{\epsilon N}{\ell\tilde{N}}\cdot\sum_{y\in Q_r}\frac{I(y)}{\tilde{N}} \leq \frac{\epsilon N}{\ell\tilde{N}}\cdot\frac{32\epsilon N}{\tilde{N}} \leq \frac{50\epsilon^2}{\ell}$$

Thus, if $\lambda_\ell\geq 100\epsilon^2/\ell$, then $\lambda_1\geq\lambda_\ell/2$ and $r=20\epsilon/mt\lambda_\ell$ suffices. ■

5 Setting the parameters

In this section we argue the claimed time-space tradeoffs using the previous analysis. For a given choice of the space parameter S, we show how to achieve parameters ℓ,m,t and r satisfying all the required constraints. Note that the actual space used by the data structure will be $\tilde{O}(mr+t+\ell)$, and we will ensure that $mr+t+\ell=O(S)$.

From the analysis in the previous section (Theorem 4.1) and the above discussion, it follows that we need to satisfy the following constraints:

1. $\ell,m,t\geq 1$. $r\geq 20\epsilon/(mt\lambda_\ell)$ if $\ell\lambda_\ell\geq 100\epsilon^2$ and $r\geq 20\epsilon N/mt$ otherwise.
2. $mt^2\lambda_\ell\leq 1/8$.
3. $\epsilon t/\ell\leq 1/4$.
4. $mr+t+\ell=O(S)$.

We first note that to satisfy these constraints, the space cannot be extremely small.

Claim 5.1 *The above constraints are feasible for a parameter S only if $S=\Omega(\sqrt{\epsilon N})$.*

Proof: For an arbitrary function f, we can always have that $\lambda_\ell = 1/N$. For a such a function, we will need to ensure that $r \geq 20\epsilon N/mt$. The lower bound now follows by noting that

$$S \cdot S \;=\; \Omega(mr) \cdot \Omega(t) \;=\; \Omega(mrt) \;=\; \Omega(\epsilon N)$$

■

We now show how to achieve the tradeoffs for different ranges of S. We assume that $S < (\epsilon N)/10$, since otherwise inverting the function on ϵ fraction of inputs is trivial using $\tilde{O}(S)$ and $\tilde{O}(1)$ time. We will also need the bound $\lambda_\ell \leq 2\epsilon/\ell$ from Claim 4.2.

Achieving $TS = O(\epsilon N)$ when $S \in (\epsilon^2 N, \epsilon N/10)$

In this case, we set the parameters as below

$$\ell = 100S, \qquad m = 8S \quad \text{and} \quad t = \frac{1}{80} \min\left\{\frac{\epsilon N}{S}, \frac{10}{\sqrt{S\lambda_\ell}}\right\}$$

We take $r = 20\epsilon/(mt\lambda_\ell)$ if $\ell \cdot \lambda_\ell \geq 100\epsilon^2$ and $20\epsilon N/mt$ otherwise. It is now easy to verify the constraints.

1. We only need to check $t \geq 1$. This follows because $\epsilon N/(80S) \geq 10$ by assumption and $1/(8\sqrt{S\lambda_\ell}) \geq 1/\sqrt{2\epsilon} \geq 1$ using Claim 4.2.

2. $mt^2\lambda_S \;\leq\; 8S \cdot \dfrac{1}{64S\lambda_\ell} \cdot \lambda_\ell \;\leq\; 1/8.$

3. $\epsilon t/\ell \;\leq\; \dfrac{\epsilon}{100S} \cdot \dfrac{\epsilon N}{80S} \;<\; \dfrac{1}{80S} \;\leq\; 1/4.$

4. $t = O(S)$ as $t \leq \epsilon N/(80S)$ and $\epsilon N = O(S^2)$. To get the bound on r, consider the following two cases:

   - If $t = O(\epsilon N/S)$, then by Claim 4.2, we can always get $r = O(\epsilon N/mt) = O(1)$ upon plugging the values of m and t.
   - Now assume $t = (1/8)\sqrt{1/S\lambda_\ell}$. Further, observe that $\ell\lambda_\ell = 100S\lambda_\ell \geq 100S/N \geq 100\epsilon^2$. The penultimate inequality uses that $\lambda_\ell \geq 1/N$ while the last one uses $S \geq \epsilon^2 N$. Hence, we can take $r = O(\epsilon/mt\lambda_\ell) = O(\epsilon/\sqrt{S\lambda_\ell})$ after plugging the values of m and t. Again using $S\lambda_\ell \geq \epsilon^2$, we get that $r = O(1)$.

   Thus we can always make r, a suitably large constant and note that $mr = O(S)$. Hence, $mr + t + \ell = O(S)$ which verifies the fourth constraint.

To verify the tradeoff, note that since $r = O(1)$, we have $T = O(t)$ and hence,

$$TS \;=\; O(m \cdot t) \;=\; O(8S \cdot (\epsilon N/S)) \;=\; O(\epsilon N).$$

Achieving $TS^3 = O(\epsilon^5 N^3)$ when $S \in (\sqrt{\epsilon N}, \epsilon^2 N)$

We divide this case into two subcases, depending on the value of $S \cdot \lambda_\ell$.

Case 1: $S \cdot \lambda_\ell \geq 100\epsilon^2$
In this case, we can in fact achieve $TS = O(\epsilon N)$ using the following parameters:

$$l = S, \qquad m = S, \qquad t = \frac{2\epsilon}{S\lambda_\ell} \quad \text{and} \quad r = \frac{20\epsilon}{mt\lambda_\ell} = 5$$

We now verify the constraints:

1. We only need to verify $t \geq 1$ which follows from $\lambda_\ell \leq 2\epsilon/S$.

2. $mt^2\lambda_\ell \;=\; S \cdot \dfrac{4\epsilon^2}{S^2\lambda_\ell^2} \cdot \lambda_\ell \;\leq\; \dfrac{4\epsilon^2}{100\epsilon^2} \;<\; \dfrac{1}{8}$.

3. $\epsilon t/\ell \;=\; \dfrac{\epsilon}{S} \cdot \dfrac{2\epsilon}{S\lambda_\ell} \;=\; \dfrac{2\epsilon^2}{S \cdot (100\epsilon^2)} \;<\; \dfrac{1}{4}$

4. We only need to verify $t = O(S)$ which follows from $t = \dfrac{2\epsilon}{S\lambda_\ell} \leq \dfrac{1}{50\epsilon} = \dfrac{S}{50} \cdot \dfrac{\epsilon N}{S^2} \cdot \dfrac{S}{\epsilon^2 N} \leq \dfrac{S}{50}$.
   The above uses that $S^2 \geq \epsilon N$ and $S \leq \epsilon^2 N$.

Since $r = O(1)$, $T = O(t)$ and $TS = O(\epsilon/\lambda_\ell) = O(\epsilon N)$ using $\lambda_\ell = \Omega(1/N)$ from Claim 4.2. Note that this also implies $TS^3 = \epsilon^5 N^3$ since $S \leq \epsilon^2 N$.

Case 2: $S \cdot \lambda_\ell < 100\epsilon^2$
In this case, we will assume that $S^3 \geq 800\epsilon^4 N^2$, since otherwise we can simply achieve the claimed tradeoff by setting $T = \epsilon N$. We choose the parameters as below:

$$\ell = S, \qquad m = \frac{S^2}{8\epsilon^2 N^2 \lambda_\ell}, \qquad t = \frac{\epsilon N}{S} \quad \text{and} \quad r = \frac{20\epsilon N}{mt}$$

It remains to verify the required conditions.

1. We only need to check $m \geq 1$. This follows from the assumptions above since
$$m \;=\; \frac{S^3}{8\epsilon^2 N^2 \cdot S\lambda_\ell} \;\geq\; \frac{S^3}{8\epsilon^2 N^2 \cdot (100\epsilon^2)} \;\geq\; 1$$

2. $mt^2\lambda_\ell \;=\; \dfrac{S^2}{8\epsilon^2 N^2 \lambda_\ell} \cdot \dfrac{\epsilon^2 N^2}{S^2} \cdot \lambda_\ell \;\leq\; 1/8$.

3. $\epsilon t/\ell \;=\; \dfrac{\epsilon}{S} \cdot \dfrac{\epsilon N}{S} \;=\; \dfrac{\epsilon^2 N}{S^2} \;\leq\; \dfrac{\epsilon^2 N}{\epsilon N} \;\leq\; 1/4$.

4. We have $t = O(S)$ since $\epsilon N = O(S^2)$. Also, note that $mr \;=\; \dfrac{20\epsilon N}{t} \;=\; 20S \;=\; O(S)$.

5. Note that in this particular case, we also need to verify that $r \geq 1$ (as in all other cases, we were setting r to be a constant). Note that because $mr = 20S$, it suffices to verify that $m \leq 20S$. To see this,
$$m = \frac{S^2}{8\epsilon^2 N^2 \lambda_\ell} \;\leq\; \frac{100S\epsilon^2}{8\epsilon^2 N^2 \lambda_\ell^2} \;<\; \frac{7S}{N^2\lambda_\ell^2} < 7S$$
   The last inequality uses $\lambda_\ell \geq 1/N$ and the first inequality uses $S \cdot \lambda_\ell < 100\epsilon^2$.

To calculate the tradeoffs, we note that for $T = tr$,

$$T = \frac{20\epsilon N}{m} = \frac{(20\epsilon N)(8\epsilon^2 N^2 \lambda_\ell)}{S^2} = \frac{(160\epsilon^3 N^3)(S\lambda_\ell)}{S^3} < \frac{16000 \cdot \epsilon^5 N^3}{S^3}$$

which proves the claim that $TS^3 = O(\epsilon^5 N^3)$.

6 Final time space tradeoffs

In this section, we write down the final time-space trade-offs that can be obtained in the RAM model for inverting functions. The following is the main theorem of this section.

Theorem 6.1 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function such that there is a data structure with parameters ℓ, m, t and r such that*

$$\mathop{\mathbb{P}}_{x \in [N]} \left[\mathsf{Invert}(f(x)) \in f^{-1}(f(x))\right] \geq \epsilon$$

where the meaning of the symbols is same as in Theorem 4.1. Then assuming that $(\ell + mr + t) = O(S)$ and $tr = O(T)$, there is an algorithm (in the RAM model) which uses space $\tilde{O}(S)$ and time $\tilde{O}(T)$ and inverts f on an ϵ fraction of inputs. Here $\tilde{O}$ hides factors of $2^{\text{poly} \log \log N}$.

Proof: We first note that Theorem 4.1 gives us that over the randomness of the data structure, the algorithm Invert inverts an ϵ fraction of the inputs. This implies that there is a fixed value of randomness (call it 'good' value) for which Invert inverts an ϵ fraction of the inputs. Note that the randomness used in the data structure is just the randomness used to sample the functions $g_1, \ldots, g_r : [N] \times [N'] \to [N]$ where $N' = O(\log^2 N)$. As these functions are t wise independent, and we pick these functions pairwise independently, the total randomness used to sample these functions is $t \cdot \log r \cdot 2^{\Theta((\log\log N + \log\log N')^3)}$ (using Corollary 8.9). This is actually just $\tilde{O}(t)$ (because $r \leq N$). Hence, we can fix the randomness to a good value using $\tilde{O}(t)$ bits.

Now, we describe the final algorithm. It is just the algorithm Invert with a fixed value of randomness. Clearly, the total space required for the algorithm is the space consumed by the data structure which is $\tilde{O}(\ell + mr)$, plus the space used to store the value of randomness which is $\tilde{O}(t)$ along with total run-time space used by the algorithm which we will see is $\tilde{O}(r + t)$. Hence, the total space used by the RAM algorithm is $\tilde{O}(mr + t + \ell) = \tilde{O}(S)$.

To find the total run time of the algorithm, note that it has three steps:

- Search in the high in-degree list
- Given the value of randomness, getting the succinct representation of g_i's
- Executing the algorithm Invert

The first step clearly takes $O(\log \ell) = \tilde{O}(1)$ time by binary search. Also, by Corollary 8.9, to get a succinct representation of the different g_i's, the total time and space required is $O(r + t)$. The final step means doing r walks of length t each to find a hit and possibly another r walks of length t to find the inverse (or find that its a false hit). Each step of the walk involves at most $2n^2 = O(\log^2 N)$ evaluations of one of the g_i's and each evaluation clearly this takes time $2^{\text{poly} \log \log N}$ (by Corollary 8.9). Thus the total time required is $\tilde{O}(rt) = \tilde{O}(T)$. We note that the only significant space consumption during execution of Invert is $O(r + t)$. Hence, we get the theorem. ■

Corollary 6.2 *For any $f : [N] \to [N]$, $\epsilon > 0$ and $S = \tilde{\Omega}(\sqrt{\epsilon N})$, there is an algorithm making oracle calls to f which inverts f on an ϵ fraction of inputs and runs in time T such that $TS^3 = \tilde{O}(\epsilon^5 N^3)$*

Proof: This is simply obtained by combining theorem 6.1 with time-space trade-offs in the second part of section 5. ■

Corollary 6.3 *For any $f : [N] \to [N]$, $\epsilon > 0$ and $S = \max\{\tilde{\Omega}(\sqrt{\epsilon N}), \tilde{\Omega}(\epsilon^2 N)\}$, there is an algorithm making oracle calls to f which inverts f on an ϵ fraction of inputs and runs in time T such that $TS = \tilde{O}(\epsilon N)$*

Proof: This is obtained by combining theorem 6.1 with time-space trade-offs in the first part of section 5. ■

Some canonical tradeoff points

We now describe some canonical trade-offs possible with our scheme. We first list down two corollaries for which it is somewhat simpler to understand the time-space trade-off curve.

Corollary 6.4 *For any $f : [N] \to [N]$, $\epsilon > 1/N^{1/5}$, there is an algorithm making oracle calls to f which inverts f on an ϵ fraction of inputs and runs in time T such that $TS^3 = \tilde{O}(\epsilon^5 N^3)$.*

Proof: This follows from corollary 6.2. In particular, for the problem to be non-trivial, as we have noted before, $S^3 \geq \epsilon^4 N^2$. However, for $\epsilon > 1/N^{1/5}$, this condition implies $S \geq \sqrt{\epsilon N}$. ■

Corollary 6.5 *For any $f : [N] \to [N]$, $\epsilon > 1/N^{1/3}$ and $S = \Omega(\epsilon^2 N)$, there is an algorithm making oracle calls to f which inverts f on an ϵ fraction of inputs and runs in time T such that $TS = \tilde{O}(\epsilon N)$.*

Proof: This follows from corollary 6.3. The condition $S = \Omega(\epsilon^2 N)$ implies $S = \Omega(\sqrt{\epsilon N})$ when $\epsilon > 1/N^{1/3}$. ■

Among interesting points that can be achieved on the curve, it follows by Corollary 6.5 that on $1/N^{1/3}$ fraction of inputs, the function f can be inverted by an oracle algorithm running in time $T = \tilde{\Omega}(N^{1/3})$ and space $S = \tilde{\Omega}(N^{1/3})$. As a generalization of the this, we now prove the following statement which was stated in the abstract.

Corollary 6.6 *For any $f : [N] \to [N]$ and $\epsilon > 0$, it is possible to get a time space trade-off scheme such that $T = S = \max\{\tilde{O}(\sqrt{\epsilon N}), \tilde{O}(\epsilon^{\frac{5}{4}} N^{\frac{3}{4}})\}$.*

Proof: Note that $\sqrt{\epsilon N} \geq \epsilon^2 N$ iff $\epsilon \leq N^{-1/3}$. Hence, in this case, we can apply corollary 6.3, to get a time space trade-off scheme with $T = \tilde{O}(\sqrt{\epsilon N}) = S$. If $\epsilon \geq N^{-1/3}$, then note that $\epsilon^{5/4} N^{3/4} \geq \sqrt{\epsilon N}$. Also observe that if we set $T = \tilde{O}(\epsilon^{5/4} N^{3/4}) = S$, then $TS^3 \geq \epsilon^5 N^3$. Hence, for $\epsilon > N^{-1/3}$, both the conditions for corollary 6.2 are satisfied if we set $S = \tilde{O}(\epsilon^{5/4} N^{3/4}) = T$ which proves the result. ■

7 Bounds based on collision probability

The analysis in the previous section might wrongly suggest that by storing a pre-computed list of points with large number of pre-images, we aim to bring down the effective collision probability of the function. This however, is a false intuition. Rather the main purpose of the pre-computed list is to put a bound on the deviation in the indegree of elements outside the high in-degree list. In fact, our analysis can be used to show that the following trade-offs can be achieved provided the function is guaranteed to be k-regular

- If $S \geq \frac{\epsilon^2 N}{k}$, then there is a time-space trade-off scheme achieving $TS = O\left(\frac{\epsilon N}{k}\right)$
- If $S < \frac{\epsilon^2 N}{k}$, then there is a time-space trade-off scheme achieving $TS^2 = O\left(\frac{\epsilon^3 N^2}{k^2}\right)$

In light of this, it is an interesting question to ask what kind of time-space trade-off schemes can be obtained if one is not allowed to use a pre-computed list. We now suggest a time-space trade-off scheme for inversion when no precomputed list of high-indegree is allowed. It is only a minor modification of the construction from the previous section.

In the data structure, we make no changes except we do not store a pre-computed list of high in-degree. In the algorithm phase, let the total time for the scheme be T. Then initially, on being given input y, the algorithm samples $T/2$ random points $x_1, \ldots, x_{T/2}$ and checks if $f(x_i) = y$ for any y. If this step fails, then the algorithm uses the algorithm of the previous section except that because there is no list of high in-degree points, the step which searches for y in the list is absent.

To analyze this scheme, we need a variant of Theorem 4.1 which we state next.

Theorem 7.1 *For given $\epsilon > 0$ and $f : [N] \to [N]$ such that the collision probability of f is λ, let the parameters m, t in the data structure be such that $mt^2\lambda < 1/8$. Then, there exists an $r = O(\epsilon N/mt)$ such that*

$$\mathop{\mathbb{P}}_{x \in [N]}\left[\mathsf{Invert}(f(x)) \in f^{-1}(f(x))\right] \;\geq\; \epsilon$$

In fact, if $\lambda > 1600\epsilon^2$, then one can take $r = O(\epsilon/mt\lambda)$

Proof: We only sketch the proof here as it is almost the same as that of Theorem 4.1. The principal difference is that since there is no pre-computed list of high in-degree elements, there is no upper bound on the size of the elements outside the (empty) list. To see the problem that can arise, assume that for a particular element y, $(tI(y))/N > 1/2$. Then Claim 4.3 fails to give a lower bound on the probability of occurrence of element y in a single walk and hence in the data structure. However, note that if $tI(y)/N > 0.1$, then it means that if we randomly sample $T/2 \geq 100t$ (we will ensure $T > 200t$; this just adds a constant overhead) points, with high probability, we will find a pre-image of y. Hence, the random sampling phase of the algorithm will invert such a y.

Now, we describe the proof of Theorem 4.1. For a particular setting of m and t, let us define $BAD := \{y : I(y) > 0.1N/t\}$. We can assume that $\sum_{y \in BAD} I(y) < 2\epsilon N$ (otherwise, the random sampling phase can invert an ϵ fraction of the elements). For elements $y \notin BAD$, Claims 4.3, 4.4, 4.5 and 4.6 all go through without any changes. Now, we define Q_r and λ_1 as in the proof of Theorem 4.1. As was shown there, if $\sum_{y \in Q_r} I(y)/N \geq 32\epsilon$, then our algorithm inverts an ϵ fraction of elements. Similarly, as was shown above, if $\sum_{y \in BAD} I(y)/N \geq 2\epsilon$, then also our algorithm inverts an ϵ fraction of the elements. Hence, we can assume that $\sum_{Q_r \cup BAD} I(y)/N < 34\epsilon$. If this is true, then as in the proof of Theorem 4.1, it is easy to show $\lambda_1 > 1/N$ and $r = O(\epsilon/mt\lambda_1)$. This gives $r = O(\epsilon N/mt)$ which is the first part of the theorem.

Also, we can prove that in this case,

$$\lambda - \lambda_1 = \sum_{y \in Q_r} \frac{I(y)^2}{N^2} + \sum_{y \in BAD} \frac{I(y)^2}{N^2} \le \left(\sum_{y \in BAD \cup Q_r} \frac{I(y)}{N} \right)^2 \le (34\epsilon)^2 < 1500\epsilon^2$$

This gives the second part of the theorem. ■

7.1 Setting the parameters

We now show how to set the parameters of the data structure when we are not allowed to use a list consisting of high in-degree elements. We first deal with the case when S is allowed to be large.

Achieving $TS = O(\epsilon N)$ when $S = \Theta(\epsilon^2 N^2 \lambda)$

As before, for convenience, we let $S \ge C\epsilon^2 N^2 \lambda$ for a very large constant C. The constant C can be made smaller by just tweaking with the constants in the parameters. Also, we let C' be a sufficiently large constant such that $S' = S/C'$. We assume that $S' \le \epsilon N$ because otherwise T is being bounded by a large constant. We now set the parameters m and t to the following values

$$m = S' \qquad t = \frac{\epsilon N}{m}$$

It is clear that both $m, t \ge 1$. Now, consider the quantity $mt^2\lambda = \epsilon^2 N^2 \lambda / m = \epsilon^2 N^2 \lambda / S' \le 1/8$ if $C \ge 8C'$. Hence, we can now apply Theorem 7.1 to get $r = O(\epsilon N/tm) = O(1)$. Hence, the total space is $mr = O(S') \le S$. Now, we set the total time $T = 200tr = O(\epsilon N/S)$ (The factor of 200 is just to account for the random sampling stage as promised in the proof of Theorem 7.1).

We also observe that $t = O(S)$ as long as $S = \Omega(\sqrt{\epsilon N})$.

Achieving $TS^2 = O(\epsilon^3 N^3 \lambda)$ when $S = o(\epsilon^2 N^2 \lambda)$

Now, we show the setting of parameters in case $S < c\epsilon^2 N^2 \lambda$ where c is a very small constant. Let $S' = S/C$ where C is a very large constant. We note that we can assume $S'^2 \ge \epsilon^2 N^2 \lambda$ as otherwise, we can make time $T = \epsilon N$ and can trivially achieve inversion of an ϵ fraction of elements. The setting of parameters is as follows:

$$m = \frac{S'^2}{\epsilon^2 N^2 \lambda} \qquad t = \frac{\epsilon N}{4S'}$$

We first check that because of the assumptions on the size of S, it clearly follows that $m, t \ge 1$. By definition of m and t, we can check that $mt^2\lambda \le \frac{1}{8}$. Hence, we can apply Theorem 7.1 to get $r = O(\epsilon N/mt) = O(\epsilon^2 N^2 \lambda / S')$. The total space required is clearly $O(mr) = O(S') \le S$. The time required is $T = 200tr = O(\epsilon^3 N^3 \lambda / S^2)$. Hence, we achieve the aforesaid trade-off *i.e.*, $TS^2 = \epsilon^3 \lambda N^3$.

We also note that in both the above scenarios, $t = O(\epsilon N/S)$ which means that if $S \ge \sqrt{\epsilon N}$, $t = O(S)$.

Remark 7.2 The trade-off given above is an analogue of the trade-off in Fiat-Naor where they show how to invert all the elements and achieve a trade-off of $TS^2 = N^3\lambda$. However, they do no put a restriction that there should be no pre-computed list of high in-degree elements.

7.2 The final time-space tradeoffs

Now, we describe the final time space trade-offs which can be obtained if we know the collision probability but are not allowed to use a high in-degree list. This is exactly analogous to Appendix 6. We simply state the main theorem here and note that the proof is identical to Theorem 6.1.

Theorem 7.3 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function which has collision probability λ such that there is a data structure with parameters m, t and r such that*

$$\mathop{\mathbb{P}}_{x \in [N]} \left[\mathsf{Invert}(f(x)) \in f^{-1}(f(x)) \right] \;\geq\; \epsilon$$

where the meaning of the symbols is same as before. Then assuming that $O(mr + t) = O(S)$ and $tr = O(T)$, there is an algorithm (in the RAM model) which uses space $\tilde{O}(S)$ and time $\tilde{O}(T)$ and inverts f on an ϵ fraction of inputs. Here $\tilde{O}$ hides factors of $2^{\mathrm{poly}\log\log N}$.

We get the following immediate corollaries of the above theorem by combining it with trade-offs derived in this section.

Corollary 7.4 *For any $f : [N] \to [N]$, $\epsilon > 0$ with collision probability λ and $S = \tilde{\Omega}(\max\{\sqrt{\epsilon N}, \epsilon^2 N^2 \lambda\})$, there is an algorithm making oracle calls to f which inverts f on an ϵ fraction of inputs and runs in time T such that $TS = \tilde{O}(\epsilon N)$*

Corollary 7.5 *For any $f : [N] \to [N]$, $\epsilon > 0$ with collision probability λ and $S = \tilde{\Omega}(\sqrt{\epsilon N})$, there is an algorithm making oracle calls to f which inverts f on an ϵ fraction of inputs and runs in time T such that $TS^2 = \tilde{O}(\lambda \epsilon^3 N^3)$*

8 Construction of k-wise independent distributions with local computability

A very important primitive required in our constructions is a family of k-wise independent functions which are formally defined below.

Definition 8.1 *A family $\mathcal{F}$ of functions $D \to R$ is said to be k-wise independent if for any $b, b_1, \ldots, b_{k-1} \in [D]$, $a, a_1, \ldots, a_{k-1} \in [R]$ satisfying $\forall\ t < k,\ b \neq b_t$, the following holds:*

$$Pr_{f \in \mathcal{F}}[f(b) = a | f(b_1) = a_1, \ldots, f(b_{k-1}) = a_{k-1}] = \frac{1}{|R|}$$

Without loss of generality, we assume that the range is $\{0,1\}^r$. Once the range is $\{0,1\}^r$, it suffices to consider family of k-wise independent functions $\mathcal{F}_1 : D \to \{0,1\}$ and take its direct product. More formally, we have the following.

Lemma 8.2 *Let $\mathcal{F}_1 : D \to \{0,1\}$ be a family of k-wise independent functions. Let $\mathcal{F}$ be defined to be a family of functions mapping D to $\{0,1\}^r$ where each $f \in \mathcal{F}$ is a r-tuple of functions from $\mathcal{F}_1$. Also, the map of f is defined as*

$$f \equiv (f_1, \ldots, f_r) : (x_1, \ldots, x_r) \mapsto (f_1(x_1), \ldots, f_r(x_r))$$

Then the family $\mathcal{F}$ is a k-wise independent. Further, if the randomness to sample an element of $\mathcal{F}_1$ be w, then the randomness to sample an element of $\mathcal{F}$ is rw. Similarly, if the time required to evaluate $f_1 \in \mathcal{F}_1$ is t, then the time required to evaluate $f \in \mathcal{F}$ is rt.

In light of the above lemma, from now on, we focus on constructing family of k-wise independent functions with a boolean range. There are many known constructions of k-wise independent functions with optimal randomness requirement (*i.e.,* $\Theta(k \log D)$) but most of them require time $\Omega(k)$ for evaluation. In contrast, we would like to have an evaluation time $(kD)^{o(1)}$ with near-optimal randomness requirement. The only one known to us which satisfies this requirement, is a construction by Ostlin and Pagh [OP03] . While the randomness required in their construction is optimal, and they also achieve a constant evaluation time in the RAM model, the construction may deviate from being uniformly random on any set of size k by up to $\frac{1}{k^c}$ for some constant c. However, an inverse polynomial error is not acceptable for our purpose.

We suggest a new construction here which does not seem to have appeared previously in literature. The idea is to use a efficiently computable lossless expander to construct k-wise independent functions. While the idea of using graphs with pseudorandom properties is present in the work by Siegel [Sie89], the construction described there is based on polynomial interpolation. In contrast, our construction is based on the following lemma :

Lemma 8.3 *Let $G = (L, R, E)$ be a d left-regular bipartite graph i.e. any vertex $l \in L$ has degree d. Also for any $V_1 \subset L$ with $|V_1| \le k$, the size of neighborhood of V_1 is bigger than $\frac{|V_1|d}{2}$. For any string $s \in \{0,1\}^{|R|}$, define $f_s : L \to \{0,1\}$ as a function mapping x to the bit-wise XOR of the bits corresponding to its neighbors in R i.e. $f_s(x) = \oplus_{i \in \Gamma(x)} s_i$. Then the family $\mathcal{F} = \{f_s : s \in \{0,1\}^{|R|}\}$ is a k-wise independent family of functions. Also if the time for evaluating a neighbor in the graph is t, then the function is computable in time $O(td)$ and the randomness required to sample a function from $\mathcal{F}$ is $|R|$.*

Proof: First, observe that the claim regarding the randomness requirement for sampling a function from $\mathcal{F}$ as well as time required to evaluate $f \in \mathcal{F}$ follow trivially. For any vertex $v \in L$, let $\Gamma(v) \in \{0,1\}^{|R|}$ denote the characteristic vector of its neighborhood. Now, note that to prove that the function family $\mathcal{F}$ is k-wise independent, it suffices to prove that for any set $V_1 \subset L$ of size at most k, the set of vectors $\{\Gamma(v)\}_{v \in V_1}$ are linearly independent.

Consider any set $V_1 \subset L$ with $|V_1| \le k$ and assume for the sake of contradiction that the vectors $\{\Gamma(v)\}_{v \in V_1}$ are linearly dependent. Consider a minimal set $A \subset V_1$ such that $\{\Gamma(v)\}_{v \in A}$ satisfy the same property. This however is same as saying that $\oplus_{v \in A} \Gamma(v) = 0$. Now, by the expansion property of G, there must be $u \in R$ such that it has exactly one neighbor in A which means that the coordinate corresponding to u in the sum $\oplus_{v \in A} \Gamma(v)$ is non-zero leading to a contradiction. Thus the family $\mathcal{F}$ is k-wise independent. ■

Thus, we are left with the task of constructing bipartite graphs which have extremely good expansion. For this, we use the construction by Capalbo *et al.* in [CRVW02]. While their main theorem gives a construction which achieves optimal expansion, it is slightly non-constructive in the sense that it requires construction of an auxillary graph which is hard to construct for extremely unbalanced expanders. We use a different result from the same paper which is more efficient to compute though slightly slack in terms of expansion.

Theorem 8.4 ([CRVW02], Theorem 7.3) *For any N and $K \le N$ and $\epsilon > 0$, there exists a D left-regular bipartite graph $G = (L, R, E)$ such that $|L| = N$, $|R| = K$ and $D = O(2^{(\frac{\log \log N}{\epsilon})^3})$ such that every set of size $K_m \le K_{max}$ has a neighborhood of size at least $DK_m(1-\epsilon)$ where $K_{max} = \Theta(\frac{\epsilon K}{D})$. Further, given $v \in L$ and $i \in D$, the i^{th} neighbor of v is computable in time poly$(\log N)$.*

For our application, we can put $\epsilon = \frac{1}{4}$ and $K = CkD$ for a sufficiently large constant C, to get the following family of expanders.

Theorem 8.5 *For any N and $k \leq N^{0.99}$, there exists a D left-regular bipartite graph $G = (L, R, E)$ such that $|L| = N$, $|R| = Ck2^{\Theta((\log\log N)^3)}$ and $D = 2^{\Theta(\log\log N)^3}$ such that every set of size $K_m \leq k$ has a neighborhood of size at least $DK_m(1-\epsilon)$. Further, given $v \in L$ and $i \in D$, the i^{th} neighbor of v is computable in time poly$(\log N)$.*

Applying Lemma 8.3, we get the following construction of family of k-wise independent functions.

Theorem 8.6 *For any $N \in \mathbb{N}$ and $k \leq N^{0.99}$, there exists an explicit construction of a family $\mathcal{F} : [N] \to \{0,1\}$ of k-wise independent functions such that the randomness required to sample $f \in \mathcal{F}$ is $k2^{\Theta((\log\log N)^3)}$. Further, for any such $f \in \mathcal{F}$, on any input x, $f(x)$ can be computed in time $2^{\Theta((\log\log N)^3)}$.*

Applying lemma 8.2, we get the following corollary.

Corollary 8.7 *For any $N, N' \in \mathbb{N}$ and $k \leq N^{0.99}$, there exists an explicit construction of a family $\mathcal{F} : [N] \times [N'] \to [N]$ of k-wise independent functions such that the randomness required to sample $f \in \mathcal{F}$ is $k2^{\Theta((\log\log N + \log\log N')^3)}$. Further, for any such $f \in \mathcal{F}$, on any input x, $f(x)$ can be computed in time $2^{\Theta((\log\log N + \log\log N')^3)}$.*

What we actually need is to sample pairwise independent functions from the family $\mathcal{F}$ above. The following fact is well known

Fact 8.8 *For any $r, m \in \mathbb{N}$, there is an algorithm running in time $O(m + r)$ such that r pairwise independent uniformly random strings can be computed using $m\lceil\log r\rceil$ bits of randomness.*

Combining the above fact and Corollary 8.7, we get the following corollary.

Corollary 8.9 *For any $N, N', r \in \mathbb{N}$ and $k \leq N^{0.99}$, there exists an explicit construction of a family G of r pairwise independent functions such that each function is a uniformly random sample of $\mathcal{F} : [N] \times [N'] \to [N]$ which is a family of k-wise independent functions. Further, the randomness required to sample these r functions is $k \cdot \log r \cdot 2^{\Theta((\log\log N + \log\log N')^3)}$. Also, given the randomness for sampling these r functions, one can get a representation of each of the r functions in total time $O(r + t)$. Also, once we have such a representation for $f \in \mathcal{F}$, for any input x, $f(x)$ can be computed in time $2^{\Theta((\log\log N + \log\log N')^3)}$.*

9 Distinguishers for Pseudorandom Generators

In this section we prove the following result

Theorem 9.1 *For every $\epsilon \leq 2^{n/2}$ and every length-increasing function $G : \{0,1\}^{n-1} \to \{0,1\}^n$, there is a circuit C of size $O(\epsilon^2 \cdot 2^n)$ such that*

$$\mathbb{P}[C(G(U_{n-1})) = 1] - \mathbb{P}[C(U_n) = 1] \geq \epsilon$$

We first show that a slightly weaker bound of $O(\epsilon^2 \cdot n \cdot 2^n)$ can be achieved via a very simple construction and analysis. Over the complete basis of fan-in two gates, the circuit has size at most $n + \epsilon^2 \cdot 2n \cdot 2^n$. The tight bound is achieved by a construction that is still rather simple, but whose analysis involves a rather technical fourth-moment calculation about the *large* deviation of four-wise independent distributions.

9.1 A Simpler Construction and Analysis

The construction in this section is based on the well known fact, which goes back at least to [AGHP92], that if a random variable X ranging over $\{0,1\}^k$ has constant statistical distance from the uniform distribution U_k over $\{0,1\}^k$, then there is a linear function that distinguishes U_k from X with advantage at least $\Omega(2^{-k/2})$. The standard proof proceeds via Fourier analysis. Here we observe a more general proof that works for any family of pairwise independent hash functions.

Lemma 9.2 *Let H be a pairwise independent family of functions $h : \Omega \to \{-1,1\}$ and $g : \Omega \to \mathbb{R}$ be any real-valued function, then there exists a function $h \in H$ such that*

$$\left|\sum_{x\in\Omega} h(x)g(x)\right| \geq \frac{1}{\sqrt{|\Omega|}} \sum_x |g(x)|$$

Proof: Consider the random variable $s_h := \sum_{x\in\Omega} h(x)g(x)$ defined over the random choices of h from H. Note that this is a sum of pairwise independent random variables and that $\mathbb{E}\, s_h = 0$, so we have

$$\begin{aligned}
\mathbb{E}\, s_h^2 &= \mathrm{V}ar\, s_h \\
&= \sum_x \mathrm{V}ar\, h(x)g(x) \\
&= \sum_x g^2(x)\, \mathrm{V}ar\, h(x) \\
&= \sum_x g^2(x)
\end{aligned}$$

This implies that there is at least one function $h_0 \in H$ such that

$$s_{h_0}^2 \geq \sum_x g^2(x)$$

and so

$$|s_{h_0}| \geq \sqrt{\sum_x g^2(x)} \geq \frac{1}{\sqrt{|\Omega|}} \sum_x |g(x)|$$

Where the last inequality is an application of CauchySchwarz. ■

Theorem 9.3 *For every $\epsilon \leq 2^{n/2}$ and every length-increasing function $G : \{0,1\}^{n-1} \to \{0,1\}^n$, there is a circuit C of size $O(\epsilon^2 \cdot 2^n)$ such that*

$$\mathbb{P}[C(G(U_{n-1})) = 1] - \mathbb{P}[C(U_n) = 1] \geq \epsilon$$

Proof: Fix $k = 2 + 2\log 1/\epsilon$, and for each $z \in \{0,1\}^{n-k}$ define the set

$$S_z := \{x \in \{0,1\}^n : z \text{ is a prefix of } x\}$$

Consider the (pairwise independent) family H of affine functions

$$h_{b,a_1,\dots,a_n}(x_1,\dots,x_n) := (-1)^{b+a_1x_1+\cdots a_nx_n}$$

Note that each function in the family is computable by a circuit of size $O(n)$. Define $p(x) := \mathbb{P}[G(U_{n-1}) = x]$ to be the probability distribution of outputs of the generator G. Note that we have

$$\sum_x \left|p(x) - 2^{-n}\right| \geq 1$$

because the left-hand side is twice the statistical distance between the output of G and the uniform distribution, which is at least $1/2$ as witnessed by the statistical test that accepts only the possible outputs of G.

For every z, by applying Lemma 9.2 to $\Omega := S_z$ and $g(x) := p(x) - \frac{1}{2^n}$, we are guaranteed the existence of a function $h^{(z)} \in H$ such that

$$\sum_{x\in S_z} h^{(z)}(x)(p(x) - 2^{-n}) \geq 2^{-k/2} \sum_{x \in S_z} |p(x) - 2^{-n}|$$

(We do not need the absolute value on the left-hand side, because $h \in H \Leftrightarrow -h \in H$.)

Now define

$$C(x) := \sum_z \mathbf{1}_{S_z}(x) h^{(z)}(x)$$

and note that C can be realized by a circuit of size at most $O(2^{n-k}\cdot n) = O(\epsilon^2 \cdot n \cdot 2^n)$. Also, observe that C is ± 1 *i.e.,* a boolean valued circuit. We have

$$\sum_x C(x)\cdot(p(x) - 2^{-n}) \geq \sum_z 2^{-\frac{k}{2}} \sum_{x\in S_z} |p(x) - 2^{-n}| \geq 2^{-\frac{k}{2}} \sum_x |p(x) - 2^{-n}| \geq 2^{-k/2}$$

that is

$$\mathop{\mathbb{E}}_{x\sim X} C(x) - \mathop{\mathbb{E}}_{x\sim U_n} C(x) \geq 2^{-k/2}$$

which is equivalent to

$$\mathop{\mathbb{P}}_{x\sim X}[C(x) = 1] - \mathop{\mathbb{P}}_{x\sim U_n}[C(x) = 1] \geq \frac{1}{2}\cdot 2^{-k/2} \geq \epsilon$$

■

9.2 The Optimal Construction

We begin by proving the following stronger form of Lemma 9.2 for four-wise independent families of functions.

Lemma 9.4 *Let H be a four-wise independent family of functions $h : \Omega \to \{-1, 1\}$ and $g : \Omega \to \mathbb{R}$ be any real-valued function, then*

$$\mathop{\mathbb{P}}_{h\sim H}\left[\left|\sum_{x\in\Omega} h(x)g(x)\right| \geq \frac{1}{3\sqrt{|\Omega|}}\sum_{x}|g(x)|\right] \geq .15$$

Proof: Consider the random variable, dependent on the choice of h from H,

$$s_h := \left|\sum_{x\in\Omega} h(x)g(x)\right|$$

Then we have

$$\mathbb{E}\, s_h^2 = \sum_x g^2(x) = ||g||_2^2$$

and

$$\mathbb{E}\, s_h^4 = 3\left(\sum_x g^2(x)\right)^2 - 2\sum_x g^2(x) = 3||g||_2^4 - 2||g||_2^2$$

so

$$\mathbb{E}(s_h^2 - 2||g||_2^2)^2 = 3||g||_2^4 - 2||g||_2^2 \leq 3||g||_2^4$$

and, using CauchySchwarz at the beginning and Markov's inequality at the end,

$$\begin{aligned}
\mathop{\mathbb{P}}_{h\sim H}\left[\left|\sum_{x\in\Omega} h(x)g(x)\right| \leq \frac{1}{3\sqrt{|\Omega|}}\sum_{x}|g(x)|\right] &\leq \mathbb{P}\left[\left|\sum_{x\in\Omega} h(x)g(x)\right| \leq \frac{1}{3}\sum_x g^2(x)\right] && (3)\\
&= \mathbb{P}\left[s_h \leq \frac{1}{3}||g||_2\right] && (4)\\
&= \mathbb{P}\left[s_h^2 \leq \frac{1}{9}||g||_2^2\right] && (5)\\
&= \mathbb{P}\left[2||g||_2^2 - s_h^2 \leq \frac{17}{9}||g||_2^2\right] && (6)\\
&= \mathbb{P}\left[\left(2||g||_2^2 - s_h^2\right)^2 \leq \left(\frac{17}{9}\right)^2 ||g||_2^4\right] && (7)\\
&\leq \frac{3||g||_2^4}{\left(\frac{17}{9}\right)^2 ||g||_2^4} && (8)\\
&= .0.8408\cdots && (9)
\end{aligned}$$

■

We can now prove our main result of this section, the existence of a generator of complexity $O(\epsilon^2 2^n)$.

Proof: [Of Theorem 9.1] Fix $k = 2\log_2 40/\epsilon$. Partition $\{0,1\}^n$ into 2^{n-k} sets S_z of size 2^k each, by defining, for every string $z \in \{0,1\}^{n-k}$, S_z to be the set of strings that have z as a prefix. Let H be an efficiently computable family of 4-wise independent functions $h : \{0,1\}^n \to \{0,1\}$. For every $y \in \{0,1\}^n$, define $g(y) := \mathbb{P}[G(x) = y] - 2^n$.

From Lemma 9.4, we have that for every set S_z

$$\mathop{\mathbb{P}}_{h\sim H}\left[\left|\sum_{x\in S_z} g(x)h(x)\right| \geq \frac{1}{3\sqrt{2^k}} \sum_{x\in S_z} |g(x)|\right] \geq .15$$

So that

$$\mathop{\mathbb{E}}_{h\sim H}\left[\sum_z \left|\sum_{x\in S_z} g(x)h(x)\right|\right] \geq .05 \cdot \frac{1}{\sqrt{2^k}} \sum_{x\in\{0,1\}^n} |g(x)| \geq .05 \cdot \frac{1}{\sqrt{2^k}}$$

where the last inequality is due to the fact that $|\sum_{x\in\{0,1\}^n} |g(x)|$ is twice the statistical distance between $G(x)$ and the uniform distribution, and that their statistical distance is at least $1/2$.

In particular, there is a function $h \in H$ such that

$$\sum_z \left|\sum_{x\in S_z} g(x)h(x)\right| \geq .05 \cdot \frac{1}{\sqrt{2^k}} \geq 2\epsilon$$

Define the function $b : \{0,1\}^{n-k} \to \{-1,1\}$ such that $b(z) = 1$ if $\sum_{x\in S_z} g(x)h(x)$ is positive, and $b(z) = -1$ otherwise. Then

$$\sum_x b(x_{|n-k})h(x)g(x) = \sum_z \left|\sum_{x\in S_z} g(x)h(x)\right| \geq 2\epsilon$$

Note that h is computable in $n^{O(1)}$ size, and b is computable in $O(2^{n-k}) = O(\epsilon^2 2^n)$ size, so that there is a circuit C of size $O(\epsilon^2 2^n + n^{O(1)})$ (giving outputs in ± 1) such that

$$\sum_x C(x)g(x) \geq 2\epsilon$$

which is the same as

$$\mathop{\mathbb{P}}_{x\in\{0,1\}^{n-1}}[C(G(x)) = 1] - \mathop{\mathbb{P}}_{x\in\{0,1\}^n}[C(x) = 1] \geq \epsilon$$

■

10 Lower Bounds

In this section we show a $T \cdot S \geq \epsilon N$ trade-off lower bound for the complexity of an inverter for one-way permutations which succeeds on an ϵ fraction of inputs. This was already established by Wee [Wee05], but here we present a slightly simpler proof based on a *randomized compression scheme* which is easier to adapt to prove the other three results in this section, which are new. Also, Wee established the optimal lower bound only when $T = \tilde{O}(\sqrt{\epsilon N})$ whereas we prove the lower bound for the full range of parameters.

- For a *family* of one-way permutations indexed by a set $[K]$ of keys, we have that either $T \geq \epsilon N$, showing the optimality of the brute-force algorithm, or $ST \geq \epsilon KN$, showing the optimality of Hellman's approach applied to the permutation $(k, x) \to (k, f(k, x))$.
- For length-increasing generator $G : [N] \to [2N]$, we show a trade-off lower bound $ST \geq \epsilon^2 N$. This is known to be tight only in the case of very small time. (Via the construction that we provide in Appendix 9.)
- For a *family* of length-increasing generators, we show that either $T \geq \epsilon^2 N$ or $ST \geq \epsilon^2 KN$.

We begin with the proof of the trade-off for lower bound for permutations. The idea of the proof is to show that if every permutation can be inverted on an ϵ fraction of inputs by an oracle algorithm that uses time T (and, in particular, makes at most T oracle queries) and space S (so that its advice is also bounded by S), then every permutation has a randomized encoding scheme that succeeds with constant probability and compresses the permutation by $S - \epsilon^2 N/T + O(\log N)$ bits. The following fact helps us to get a lower bound on T and S from such a probabilistic encoding.

Fact 10.1 *Suppose there is a randomized encoding procedure $Enc : \{0,1\}^N \times \{0,1\}^r \to \{0,1\}^m$ and a decoding procedure $Dec : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^N$ such that*

$$\mathop{\mathbb{P}}_{r \in U_r}[Dec(Enc(x,r), r) = x] \geq \delta$$

Then $m \geq N - \log 1/\delta$.

Proof: By a standard averaging argument, we get that there is a r such that for at least a δ fraction of the x's, $Dec(Enc(x,r), r) = x$. However, that means that $Enc(x,r)$ must attain at least $\delta 2^N$ values as x varies over $\{0,1\}^N$. As the total number of values that $Enc(x,r)$ can take is bounded by 2^m, $2^m \leq \delta 2^N$, thus giving us the required inequality. ■

Using the above fact and that the encoding scheme succeeds with constant probability, we get $S \geq \epsilon^2 N/T - O(\log N)$.

Theorem 10.2 *Fix an oracle algorithm A which makes at most T oracle queries and which takes an advice string of length S. Fix a parameter ϵ.*

There are randomized encoding and decoding procedures E, D which use shared randomness and such that if f is a permutation and adv is an advice string such that

$$\mathbb{P}[A^f_{adv}(f(x)) = x] \geq \epsilon$$

then

$$\mathop{\mathbb{P}}_{r}[D(r,\ E(r,f)) = f] \geq .9$$

and the length of $E(R,f)$ is at most

$$\log N! - \frac{\epsilon N}{100T} + S + O(\log N)$$

bits.

Proof: We use the shared randomness r to generate a random subset $R \subseteq [N]$ such that each element of $[N]$ is independently chosen to be in R with probability $1/10T$.

We say that an element $x \in R$ is *good* if: (i) $A^f_{adv}(f(x)) = x$ and (ii) none of the oracle queries in the computation $A^f_{adv}(f(x))$ are in R, except possibly for the query x. Let G be the set of good elements of R.

We claim that, with probability at least .9 over the choice of R, $|G| \geq \epsilon N/100T$.

To prove the claim, note that, for a fixed $x \in [N]$,

- the probability that x is in R is exactly $1/10T$, and
- the probability that all queries (except possibly x) in the computation of $A^f_{adv}(f(x))$ are outside of R is at least $1 - (1 - 1/10T)^T \approx 1 - e^{-10}$, and it is more than $1 - 1/100$ for large enough T.

Note also that the two events are independent, so the average number of x in R such that A inverts $f(x)$ but makes queries inside of R is at most $I/1,000T$, where I is the number of elements of $[N]$ which are inverted by A. By Markov's inequality, with probability at least .95 this quantity is at most $I/50T$, and by Chernoff bound the probability that R contains at least $I/20T$ elements of $[N]$ that A can invert is also at least .95, and so with probability at least .9, R contains at least $3I/100T \geq \epsilon N/100T$ good elements, proving the claim.

From now we describe the encoding assuming $|G| \geq \epsilon N/100T$.

The encoding contains the following information:

- The advice string adv
- The cardinality of the set G of good elements of R
- The set $f(R)$, encoded using $\log \binom{N}{|R|}$ bits
- The values of f restricted to $f : [N] - R \to [N] - f(R)$, encoded using $\log(N - |R|)!$ bits.[6]
- The set $f(G)$ of images of good elements of R, encoded using $\log \binom{|R|}{|G|}$ bits
- The values of f restricted to $f : R - G \to f(R - G)$, encoded using $\log(|R| - |G|)!$ bits.[7]

The decoding proceeds as follows: it initializes an empty table to store the values of f, and it fills up the mapping from $[N] - R$ to $[N] - f(R)$. Next, for every element $y \in f(G)$, it finds its inverse (because all oracle queries can be answered). At this point, we know the set G as well as the value of f on every point in $([N] - R) \cup G$. To compute f on $R - G$, note that we (now) know G, R, the set $f(R - G)$ as well as the permutation restricted to $R - G$. Hence, we can compute f on the remaining points *i.e.,* $R - G$. This describes a complete decoding procedure for f. All that remains is to compute the length of the encoding.

[6]That is, this part of the encoding is a permutation $g : [N - |R|] \to [N - |R|]$, with the meaning that if $g(i) = j$, then f maps the i-th element of the set $[N] - R$ to the j-th element of the set $[N] - f(R)$. Note that knowledge of the sets R and $f(R)$ is needed to decode this part of the encoding. This will not be a problem because the decoder knows R, which is part of the common random string, and is given $f(R)$.

[7]Similar remarks hold as we made in the previous footnote. The decoder needs to know the sets $R - G$ and $f(R - G)$ to decode this part of the encoding. Although we have not explicitly specified the set G, it is possible for decoder to reconstruct G from the encoding. See the description of the decoding procedure below.

$$S + \log\left(\frac{N!}{(N-|R|)!|R|!} \cdot (N-|R|)! \cdot \frac{|R|!}{(|R|-|G|)!|G|!} \cdot (|R|-|G|)!\right) + O(logN)$$
$$= S + \log N! - \log |G|! + O(\log N)$$

■

Corollary 10.3 *If A is an oracle algorithm that runs in time at most T and such that for every permutation f there is a data structure adv of size $\leq S$ such that*

$$\mathop{\mathbb{P}}_x[A^f_{adv}(f(x)) = x] \geq \epsilon$$

Then

$$S \cdot T = \tilde{\Omega}(\epsilon N)$$

Proof: Since the randomized encoding procedure compresses representation by $S - (\epsilon N/100T) - O(\log N)$ bits and succeeds with constant probability, using fact 10.1 we get,

$$S - (\epsilon N/100T) - O(\log N) < 1 \qquad \Rightarrow \qquad ST = \tilde{\Omega}(\epsilon N)$$

■

To prove lower bounds on pseudorandom generators, we first prove the following lemma. A special case of this lemma (for $\epsilon = 1/2$) was proven by Yao [Yao90].

Lemma 10.4 *Fix an oracle algorithm A which makes at most T oracle queries, which is not allowed to query its input to the oracle, and which takes an advice string of length S. Fix a parameter ϵ.*

There are randomized encoding and decoding procedures E, D which use shared randomness such that if $p : [N] \to \{0,1\}$ satisfies

$$\mathbb{P}[A^p_{adv}(x) = p(x)] \geq \frac{1}{2} + \epsilon$$

Then

$$\mathop{\mathbb{P}}_r[D(r,\ E(r,p)) = p] \geq \Omega(\epsilon/T)$$

and the length of $E(r,p)$ is always at most

$$N - \frac{\epsilon^2 N}{10T} + S + O(1)$$

Proof: We first modify A so that it makes exactly T distinct queries while never querying its input. Clearly, the same success probability can be maintained by simply ignoring the oracle's answers on the extra queries.

Pick a random subset $R \subseteq [N]$ by independently placing each element of $[N]$ in R with probability $1/10T$. An element $x \in R$ is *good* if the computation $A^f_{adv}(x)$ makes no query inside R. Let G be the set of good elements. Let G_0 be the set of elements of G on which A is correct and G_1 the set on which it is incorrect. We say that R is good if

$$|G_0| - |G_1| \geq \frac{\epsilon N}{20T} \qquad \text{and} \qquad |G| = \Omega\left(\frac{N}{T}\right)$$

We prove that R is good with probability at least $\epsilon/20T$, and that given a good R we can achieve the required compression.

Note that for each x we have

$$\mathbb{P}[x \in G] = \left(1 - \frac{1}{10T}\right)^T \cdot \frac{1}{10T}$$

Then the average, over the choice of R of the difference between $|G_0|$ and $|G_1|$ is

$$\mathop{\mathbb{E}}_R[|G_0| - |G_1|] \geq \left(1 - \frac{1}{10T}\right)^T \cdot \frac{1}{10T} \cdot 2 \cdot \epsilon \cdot N \geq \frac{\epsilon N}{10T}$$

On the other hand, for every choice of R we have $|G_0| - |G_1| \leq N$, so with probability at least $\epsilon/20T$ we have

$$|G_0| - |G_1| \geq \epsilon N/20T$$

Further, note that $\mathbb{E}_R[|G|] = \Theta(N/T)$. Hence, a simple application of the Chernoff bound gives us that

$$\mathop{\mathbb{P}}_R[|G| = O(N/T)] \geq 1 - e^{-\frac{2N}{T}}$$

Hence, we get that R is good with probability at least $\epsilon/20T - \exp(-2N/T) \geq \epsilon/40T$. To see this, clearly $T \leq \epsilon N$ to prove any non-trivial result while we can safely assume that $T \geq N/\log N$. Assuming now that we have a good R, the encoding contains

- p restricted to $[N] - R$, taking $N - |R|$ bits;
- p restricted to $R - G$, taking $|R| - |G|$ bits; note that knowledge of the set G is needed to decode this information, but given R and the values of f outside R then the set G is completely specified.
- the set G_0, taking $|G| \cdot \mathbb{H}(1/2 + \epsilon N/40T|G|)$ bits, which is at most $|G| - \frac{\epsilon^2 N^2}{O(T^2|G|)}$

We now explain briefly how the decoding algorithm works. It initializes an empty table for p, and then fills all the entries corresponding to $[N] - R$. Next it computes G as follows: For every $x \in R$, it runs the algorithm A. If A makes any query outside R, we already know the answer and hence A can continue. If A makes any queries outside R, then $x \notin G$. Clearly, this test is both sound and complete. Now, that we know G, we know $R - G$ and hence the value of p can be computed on $R - G$ as well. The only remaining part of the reconstruction is computation of p on G. Since, by definition G makes queries only outside R, clearly, we can compute $A(x)$ for $x \in G$. Subsequently, we flip all the answers on $G - G_0$ to get the complete truth table of p. Overall the length of the encoding is

$$S + N + O(1) - \epsilon^2 N^2/T^2 O(|G|) \leq S + N + O(1) - \Omega(\epsilon^2 N/T)$$

because $|G| = O(N/T)$ ■

Theorem 10.5 *Suppose that A is an oracle algorithm that makes T queries, uses a S-bit advice string, and is such that for every length-increasing function $G : [N] \to [N] \times \{0,1\}$ there is an advice string adv such that*

$$|\,\mathbb{P}[A^G_{adv}(G(x)) = 1] - \mathbb{P}[A^G_{adv}(y) = 1]| \geq \epsilon$$

Then $S \cdot T \geq \tilde{\Omega}(\epsilon^2 N)$

Proof: Consider the set of $2^N \cdot N!$ pairs f, p where $f : [N] \to [N]$ is a permutation and p is a predicate. We show that (f, p) can be encoded using less than $\log N! + N + S + O(1) - \Omega(\epsilon^2 N/T)$ bits. To see why proving this suffices, note that we achieve a total compression of $\Omega(\epsilon^2 N/T) - S - O(1)$ which should be at most 0 implying that $S \cdot T \geq \tilde{\Omega}(\epsilon^2 N)$.

To get the encoding, we first apply Yao's reduction of distinguishers to predictors [Yao82] to obtain an algorithm B that uses at most $S + 1$ bits of advice and such that

$$P[B^G_{adv}(f(x)) = p(x)] \geq \frac{1}{2} + \epsilon$$

Clearly, one of the following two cases must be true:

- With probability at least $\epsilon/2$, $B^G_{adv}(f(x))$ queries x. We can give the entire truth table of p as advice by adding N additional random bits, and then this gives us a circuit which queries x, and hence inverts $f(x)$ on at least an $\epsilon/2$ fraction of the inputs. By Theorem 10.2, this implies that there is a randomized encoding for f using $\log N! + S + O(1) - \Omega(\epsilon N/T)$ bits. Since, we also have an encoding of p, this gives an encoding of $(f \circ p)$ using $\log N! + N + S + O(1) - \Omega(\epsilon^2 N/T)$ bits.

- Otherwise, we give the entire truth table of f as advice to B_{adv} using an additional $\log N!$ bits. Because B_{adv} has the truth table of f, we can interpret B_{adv} as a circuit, which satisfies

  $$P[B^p_{adv}(x) = p(x)] \geq \frac{1}{2} + \epsilon$$

  with the additional property that it does not query x on at least a $1 - \epsilon/2$ fraction of the inputs. Now, we construct a modified circuit C_{adv} which is same as B_{adv} except whenever B_{adv} queries p on x, C_{adv} outputs a random answer. Then we get C_{adv}

  $$P[C^p_{adv}(x) = p(x)] \geq \frac{1}{2} + \frac{\epsilon}{2}$$

  with the additional property that C_{adv} never queries its inputs. We now use Lemma 10.4 to get a randomized encoding of $(f \circ p)$ with $N + S + \log N! + O(1) - \Omega(\epsilon^2 N/T)$

  In either of the cases, the encoding succeeds with probability at least ϵ/T, we get that $S + \log(\epsilon/T) \geq \Omega(\epsilon^2 N/T) - O(1)$ and hence $S \cdot T \geq \tilde{\Omega}(\epsilon^2 N)$.

■

We say that a function $f : [K] \times [N] \to [N]$ is a *family of permutations* if, for every $k \in [K]$, the mapping $x \to f(k, x)$ is a permutation.

Lemma 10.6 *Let A be a non-uniform oracle algorithm that takes S bits of advice and makes at most T oracle queries. Then there are randomized encoding and decoding procedures such that for every family of permutations $f : [K] \times [N] \to [N]$ such that there is an advice string adv such that*

$$\mathop{\mathbb{P}}_{k \in [K],\ x \in [N]} [A^f_{adv}(k, f(k,x)) = x] \geq \epsilon$$

The encoding is decodable with probability at least .9 and has length at most $K \log N! + S + O(K \log N) - K\epsilon N/T$.

Proof: Choose a subset $R \subseteq [K] \times [N]$ by selecting each element independently with probability $\epsilon N/10T$. We say that an element $(k,x) \in R$ is *good* if $A^f(k, f(k,x))$ makes oracle queries outside of $R - \{(k,x)\}$ and successfully inverts $f(k,x)$. Let G be the set of good elements. Then, as proved before in Theorem 10.2, with probability at least .9 the set G has cardinality at least $\epsilon KN/100T$. The description of the family of permutations is given by providing, for every k:

- The advice string adv
- The cardinality of the set G_k for each k
- The set $f^{-1}(R_k)$, where $R_k := \{y : (k,y) \in R\}$
- The set $f^{-1}(G_k)$, where $G_k := \{y : (k,y) \in G\}$
- The inverse of f on $\{k\} \times ([N] - R_k)$
- The inverse of f on $\{k\} \times (R_k - G_k)$

We note that the above description is basically just providing the description for reconstructing the permutation restricted to each member of the family. The reconstruction follows exactly the same steps as in Theorem 10.2. By previous calculation, this has length

$$\sum_k \log N! + O(K \log N) + S - \sum_k \log |G_k|! \leq \log KN! + O(K \log N) + S - |G|$$

Assuming R is good, the above quantity is bounded by $\log KN! + O(K \log N) + S - K\epsilon N/T$. ■

Corollary 10.7 *Suppose that A is an oracle algorithm that makes T queries, uses an S-bit advice string, and is such that for every family of permutations $f : [K] \times [N] \to [N]$ there is an advice string adv such that*

$$\mathop{\mathbb{P}}_{k \in [K],\ x \in [N]} [A^f_{adv}(k, f(k,x)) = x] \geq \epsilon$$

Then $S \cdot T \geq \tilde{\Omega}(\epsilon KN) - \tilde{O}(KT)$

Proof: Using Lemma 10.6, we get a total compression of $K\epsilon N/T - O(K \log N)$ with probability at least 0.9. Using Fact 10.1, immediately gives us

$$S \geq K\epsilon N/T - O(K \log N) - O(1) \qquad \Rightarrow \qquad S \cdot T \geq \tilde{\Omega}(\epsilon KN) - \tilde{O}(KT)$$

■

Theorem 10.8 *Suppose that A is an oracle algorithm that makes T queries, uses an S-bit advice string, and is such that for every family of length-increasing functions $G : [K] \times [N] \to [N] \times \{0,1\}$ there is an advice string adv such that*

$$|\underset{x\in[N]}{\mathbb{P}}[A^G_{adv}(k, G(k,x)) = 1] - \underset{y\in[2N]}{\mathbb{P}}[A^G_{adv}(k,y) = 1]| \geq \epsilon$$

Then $S \cdot T \geq \tilde{\Omega}(\epsilon^2 KN) - \tilde{O}(KT)$, that is,

$$T \geq \tilde{\Omega} \min \left\{ \epsilon^2 N,\ \frac{\epsilon^2 KN}{S} \right\}$$

Proof: Let $f : [K] \times [N] \to [N]$ be a family of permutations and $p : [K] \times [N] \to \{0,1\}$ be a family of predicates. Consider the family of length-increasing functions $G(k,x) := (f(k,x), p(k,x))$. By Yao's equivalence of indistinguishability and predictability we have an algorithm B using $S+1$ bits of advice such that

$$\mathbb{P}[B^{f,p}_{adv}(k, f(k,x)) = p(k,x)] \geq \frac{1}{2} + \epsilon$$

We distinguish two cases. If B queries (k,x) for at least an $\epsilon/2$ fraction of the inputs (k,x), then we add the entire truth table of p as advice. Note that now B can be seen as inverting a family of permutations on an $\epsilon/2$ fraction of the inputs. Consequently, we can use the probabilistic encoding given in Lemma 10.6 and get a probabilistic encoding of the pair (f,p) which achieves a compression of $\epsilon NK/T - O(K \log N)$. In case, B queries (k,x) on less than $\epsilon/2$ fraction of the inputs, we do the following. We first give the entire truth table of f as advice to B. Then B can be seen as a circuit computing p on at least $1/2 + \epsilon$ fraction of inputs and making queries to p on (k,x) on at most an $\epsilon/2$ fraction of the inputs. From this, we consider the modified circuit C which behaves the same as B except on inputs where B queries p on the input, C just makes a random guess. It is clear that C satisfies

$$\mathbb{P}[C^p(k,x) = p(k,x)] \geq \frac{1}{2} + \frac{\epsilon}{2}$$

We sample a subset $R \subseteq [K] \times [N]$ by picking each element with probability $1/10T$. We say that an element (k,x) is good if C makes queries outside R. We let G_0 be the set of good elements on which C is correct and G_1 the set of good elements on which it is incorrect. As in Lemma 10.4, it can shown that with probability at least $(\epsilon/100T)$ we have

$$|G_0| - |G_1| \geq \epsilon KN/100T \qquad \text{and} \qquad |G_k| \leq \frac{N}{T}$$

where G_k denotes $G \cap \{k\} \times [N]$. Now assume that, we have a R which is good. With this, we encode p with the following bits of information:

- For all k, $p(k,x)$ such that $x \in N - R_k$ where $R_k := \{x : (k,x) \in R\}$
- For all k, $p(k,x)$ such that $x \in R_k - G_k$
- The set G_{0k} where $G_{0k} = G_0 \cap \{k\} \times [N]$.

To do the reconstruction, we apply the decoding procedure used in Lemma 10.4 restricted to $[N], R_k, G_k$. We do not re-describe it here as it is identical. Let d_k denote

$$d_k := |G_{0,k}| - |G_{1,k}|$$

Then, the calculation in Lemma 10.4 shows that for each k, we achieve a compression of $d_k^2/|G_k|$. Also, note that $\sum_k d_k = \epsilon KN/100T$. Hence, the total compression is

$$\sum_k d_k^2/|G_k| \geq \sum_k d_k^2/(N/T) \geq \frac{T}{KN}(\sum_k d_k)^2 = \epsilon^2 KN/10,000T$$

bits. Here the first inequality uses that $G_k \leq N/T$ and the second is an application of Cauchy-Schwarz inequality. As the compression succeeds with probability ϵ/T and compresses at least $\Omega(\epsilon^2 NK/T)$ bits in one case and $\epsilon NK/T - O(K \log N)$ in the other, we get that $S \geq \Omega(\epsilon^2 NK/T) - O(K \log N) - \log(T/\epsilon)$ giving us the final result (We use that $\log(T/\epsilon) = O(\log N)$.)

■

Acknowledgements

We would like to thank Daniel Wichs for suggesting the study of the common random string model, and Cynthia Dwork, Omer Reingold, Udi Wieder and Hoeteck Wee for pointers to the literature.

References

[ACR97] Alexander E. Andreev, Andrea E.F. Clementi, and José D.P. Rolim. Optimal bounds for the approximation of boolean functions and some applications. *Theoretical Computer Science*, 180:243–268, 1997. 2, 8

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992. 2, 7, 8, 29

[BBBV97] Charles Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997. 3

[BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Proceedings of CRYPTO'06*, pages 1–21, 2006. 2

[CRVW02] Michael R. Capalbo, Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 659–668, 2002. 10, 11, 27

[FN99] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM Journal on Computing*, 29(3):790–803, 1999. 1, 2, 4, 10, 11, 13

[Gol09] Alexander Golynski. Cell probe lower bounds for succinct data structures. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 625–634, 2009. 1

[Gro96] Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 212–219, 1996. 3

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, pages 305–313, 2000. 8

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401 – 406, 1980. 1, 4

[OP03] Anna Ostlin and Rasmus Pagh. Uniform hashing in constant time and linear space. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 622–628, 2003. 6, 9, 27

[Sie89] Alan Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 20–25, 1989. 27

[Wee05] Hoeteck Wee. On obfuscating point functions. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 523–532, 2005. 8, 32

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982. 37

[Yao90] Andrew Yao. Coherent functions and program checkers. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pages 84–94, 1990. 1, 8, 35

ECCC ISSN 1433-8092
http://eccc.hpi-web.de