Fast Pseudorandomness for Independence and Load Balancing*
(Extended Abstract)

Raghu Meka[1], Omer Reingold[1], Guy N. Rothblum[1], and Ron D. Rothblum[2]

[1] Microsoft Research
{meka,omer.reingold}@microsoft.com, rothblum@alum.mit.edu
[2] Weizmann Institute of Science
ron.rothblum@weizmann.ac.il

Abstract. We provide new constructions of several fundamental pseudorandom objects. Loosely speaking, these constructions obtain exponential improvements in efficiency compared to previous constructions with comparable randomness complexity. Our measure of efficiency is the number of word operations, as captured by the well-established unit-cost word RAM model. Our main results are the following:

1. A family of $(1/n)$-almost $\log n$-wise independent Boolean hash functions with $O(\log n)$ description length (or seed length) and $O(\log \log n)$ operations per evaluation.
   Prior constructions with similar seed lengths required $\Theta(\log n)$ operations.
2. ε-biased sequences for $\varepsilon = 1/\mathrm{poly}(n)$ with seed length $O(\log n \log \log n)$ and $O((\log \log n)^2)$ operations (to evaluate an output bit or a block of up to $\log n$ consecutive bits).
   Prior constructions achieve $O(\log n)$ seed length, but require $\Theta(\log n)$ operations. This construction implies pseudorandom generators with similar efficiency that fool classes such as low-degree polynomials and read-once CNFs.
3. Hash functions for placing n balls in n bins such that with all but probability $1/n$ the maximal load is $O(\log n/\log \log n)$ (which is optimal), with seed-length $O(\log n \log \log n)$ and $O((\log \log n)^2)$ operations per evaluation.
   The previously known construction with similar seed length required $\Theta(\log n \log \log n)$ operations. Indeed, our construction is an efficient instantiation of that construction, due to Celis, Reingold, Segev and Wieder (FOCS 2011).

These constructions are all simultaneously within $\log \log n$ factors of the optimal seed length, and within $(\log \log n)^2$ factors of the optimal computational efficiency.

1 Introduction

Randomness is a valuable resource in the theory and the practice of computing. Designs of randomized data structures, algorithms, and protocols often assume

* The full version is available on the authors' homepages.

J. Esparza et al. (Eds.): ICALP 2014, Part I, LNCS 8572, pp. 859–870, 2014.

access to a truly random object, such as a random function or a sequence of random bits. In many settings, however, these random objects must be replaced with a succinct and efficiently computable explicit construction that mimics some of their properties. For example, a pseudorandom bit generator, or a small family of hash functions.

In this work, we are interested in fast and efficient constructions of pseudorandom objects. Motivated by applications to data structures and algorithms, we aim for efficiency in the *(unit-cost) word RAM model*. This model measures complexity in terms of *word operations*, and is a (perhaps *the*) central model in the theory and practice of data structure and algorithm design. Applications aside, the word-RAM complexity of pseudorandom objects is a foundational question, and it has been explored in an important body of work (e.g. [NN93, Sie04, CRSW13, Tho13]). We focus on three fundamental pseudorandom objects: almost-independent hash functions, small-bias generators, and load balancing hash functions. Loosely speaking, we obtain exponential efficiency improvements compared to the fastest known constructions for each of these objects, while maintaining (up to $\log \log$ factors) the best seed length. We proceed with an overview of our contributions and their relationship to prior works. A more detailed discussion of the RAM model follows. We note that these results have already found applications in subsequent recent works of Reingold, Rothblum and Weider [RRW14] and of Reingold and Vardi [RV14].

Limited Independence. Families of hash functions with limited independence are central objects in the study of hashing and derandomization, and have many applications to algorithms and data structures. Limited independence suffices in many cases where fully random hash functions are used, and functions with limited independence have the advantage of a much more succinct and space-efficient representation. A hash family is said to be ε-almost k-wise independent if for any k fixed inputs, their (joint) output distribution is ε-close to uniform (in statistical distance, where the probability is over the choice of a function from the family). The use of limited independence in computer science begins with the seminal work of Carter and Wegman [CW79, WC81]. See also the survey of Luby and Wigderson [LW05].

A particularly interesting parameter choice is that of $(1/\mathrm{poly}(n))$-almost $\log n$-wise independent Boolean hash functions, which lend themselves to showing concentration bounds with polynomially small errors. For this setting of parameters, known constructions achieve optimal seed length of $O(\log n)$ bits [NN93], but computing these functions is expensive, and requires $O(\log n)$ word operations (where the word length is $O(\log n)$). Alternatively, Siegel [Sie04] and Thorup [Tho13] propose functions with $n^{\Omega(1)}$-independence that can be evaluated in a *constant* number of word operations, but use storage or key length $n^{\Omega(1)}$ (for functions with constant evaluation time, larger key length or storage is essential, even for logarithmic output length and independence, by Siegel's lower bound [Sie04]). Our first result is a new construction of *Boolean* hash functions with logarithmic seed length/storage and fast evaluation.

Construction 1 (Fast Almost Independent Hash Family). *We construct a family of* $(1/\mathrm{poly}(n))$*-almost* $\log n$*-wise independent functions from* $\{0,1\}^w$ *to* $\{0,1\}$*, with a* $O(\log n)$*-bit seed, which can be evaluated in* $O(\log\log n)$ *word operations.*

Moreover, for any $m \in [\log n]$*, we construct a* $(1/\mathrm{poly}(n))$*-almost* $((\log n)/m)$*-wise independent family of hash functions from* $\{0,1\}^w$ *to* $\{0,1\}^m$*, with a* $O(\log n)$*-bit seed, which can be evaluated in* $O(\log\log n)$ *operations.*

This construction follows from a new fast small-bias generator described below. It achieves an exponential efficiency improvement over past work, while still having optimal seed length / storage (up to constant factors). Moreover, it also allows fast computation of m-bit outputs: as the output size increases to $m > 1$ bits, the independence guarantee decreases proportionally to $(\log n)/m$.

It is interesting to compare our upper bound with the lower bound of Siegel [Sie04] for computing k-wise independent hash functions. Siegel's seminal *cell-probe* lower bound shows that when hashing from a universe of size n to a range of size 2^m with m-bit long *cells*, to get k-wise independent hash functions that can be evaluated in time $t < k$ (t here is the number of cell probes), one needs storage or key length $\Omega(k \cdot n^{1/t})$ (this is the length of the hash key that allows computation in t cell probes). Adapted to the RAM model, Siegel's lower bound says that to get $\log n$-wise independent hash functions with $\log n$ output length, even using $\log^\gamma n$ time, requires large $\Omega(2^{\log^{1-\gamma} n})$ storage. In contrast, we focus on *Boolean* functions, and obtain both logarithmic seed length/storage and $O(\log\log n)$ time.

Small-Bias Generators. A pseudorandom generator maps a short random seed into a longer output that is indistinguishable from the uniform distribution to a certain class of tests (or distinguishers). A small-bias generator is one that "fools" linear tests. Namely, G is an ε-biased generator if for every non-zero test vector $\mathbf{t}$, it holds that:

$$\Pr_{seed}[\langle \mathbf{t}, G(seed)\rangle = 0] \in [1/2 - \varepsilon, 1/2 + \varepsilon] \tag{1.1}$$

(the inner product is taken over $\mathbb{GF}(2)$). The notion of ε-biased generators was introduced by the seminal work of [NN93], who also gave the first constructions of such generators. Since then, a rich sequence of works provided alternative constructions and related notions [AGHP92, AIK$^+$90, RSW93], [AMN98, EGL$^+$98, AM95, BATS13]. Small-bias generators have found numerous applications throughout theoretical computer science, from derandomization [NN93], to learning theory [AM95], to efficient low-degree tests and short PCPs [BFLS91, FGL$^+$96, BSGH$^+$06]. In coding theoretic terms, they are equivalent to linear error correcting codes over $\mathbb{GF}(2)$ where all codewords have relative Hamming weight between $(1-\varepsilon)/2$ and $(1+\varepsilon)/2$ [NN93, AGHP92]. They are also an important tool in finding explicit constructions, e.g. for graphs [Nao92, AR94, MW04], two-source extractors [Raz05], and psuedorandom generators that fool low degree polynomials [BV10, Lov09, Vio09] or Read-Once CNFs [GMR$^+$12], and many more.

One especially important motivation for the study of small-bias generators, is their application to the construction of almost k-wise independent hash functions. For this application, [NN93] introduced the relaxed notion of (k, ε)-biased generators, which are only required to fool linear tests of weight at most k (i.e. vectors with at most k non-zero coordinates). Any (k, ε)-biased generator immediately implies an $(\varepsilon \cdot 2^{k/2})$-almost k-wise independent Boolean hash family. Thus, to construct good almost k-wise independent hash functions, we need a (k, ε)-biased generator with bias smaller than $2^{-k/2}$. Indeed, Construction 1 above is obtained by building a new fast $(\log n, \mathrm{poly}(1/n))$-biased generator, whose output bits (and blocks) can be computed in few word operations.

Known constructions of ε-biased generators achieve asymptotically optimal seed length of $O(\log(n/\varepsilon))$ up to constant factors, [NN93, AGHP92]. Moreover, [NN93] gave a generator with constant bias, where each of the output bits could be computed using $O(1)$ word operations. However, for smaller biases, e.g. bias $\mathrm{poly}(1/n)$ (which is necessary for obtaining almost $\log n$-wise independent hash functions), known construction are not nearly so efficient. In particular, they require $O(\log n)$ word operations for computing each output bit. This is true even for the relaxed requirement of $(\log n, \varepsilon)$-bias.

Construction 2 (ε-Biased Generator). *We construct a $(1/\mathrm{poly}(n))$-biased generator G with seed length $O(\log n \log\log n)$, where each bit of G's output can be computed in $O((\log\log n)^2)$ word operations.*

Construction 2 gives an exponential efficiency improvement over prior constructions, at the cost of a $\log\log n$ blowup in the seed length. This is our most technically elaborate construction. It utilizes almost $\log n$-wise independent hash families (see above), and load balancing hash families (see below).

Significant attention has been devoted to the circuit or arithmetic complexity of computing ε-biased generators, yielding generators computable in circuit classes such as NC^0, $\mathrm{AC}^0[\oplus]$, or low-degree polynomials [CM01, GV04, AIK06, MST06, HV06, Hea08, Shp09]. These works, like ours, contend with the fundamental question of efficiently computing pseudorandom objects. Our work, however, focuses on complexity in terms of running time in the word-RAM model, a very different model (which is arguably more relevant for many applications and prevalent architectures). Indeed, as noted above, the original work of Naor and Naor [NN93] also considered the word-RAM model (see above). Another difference is in showing that word RAM running-time measure allows for efficient constructions of a rich variety of pseudorandom objects. This is not the case if we restrict locality or algebraic degree: for example, no generator of constant degree or locality can have super-polynomial stretch [MST06].

Given the ubiquitous nature of ε-biased generators and almost k-wise independent hash functions, we get similar improvements for several constructions (see the full version for details):

- Efficient Pseudorandom Generators (PRGs) for low-degree polynomials over $\mathbb{GF}(2)$: The works of [BV10, Lov09, Vio09] show that sum of small-bias generators fool low-degree polynomials over $\mathbb{GF}(2)$. As a result we get very efficient PRGs for this well-studied class of test functions.

– Efficient PRGs for Read-Once CNFs: The work of [GMR+12] uses small-bias generators to fool read-once CNFs and hence we get very efficient PRGs for read-once CNFs.

Load Balancing. A fundamental fact in the analysis of randomized algorithms is that if n items are hashed into n bins, using a truly random hash function, then with high probability each bin contains at most $O(\log n/\log\log n)$ items. It is natural to try and construct succinct explicit functions that share this important property. It is known that $O(\log n/\log\log n)$-wise independent hash families (with $\log n$ output bits) have this property, but these require a large $O(\log^2 n/\log\log n)$-bit seed, and evaluation takes $O(\log n/\log\log n)$ word operations. Until recently no constructions with smaller seed length were known (variants of this question were posed in [ADM+99, PPR07]). In recent work, Celis *et al.* [CRSW13] construct a family with seed length $O(\log n\log\log n)$, obtaining the first improvement over the seed length of a generic $\log n$-wise independent hash family. That construction did not improve the evaluation time, but they also present a second construction, which supports evaluation in $\sqrt{\log n}$ operations. While the seed length of this second construction increased to $\log^{3/2} n$, it was the first construction with a sub-polynomial seed that beat the evaluation time of $O(\log n)$. Indeed, in light of Siegel's lower bound [Sie04], this second construction gives a separation between the number of operations needed to compute load balancing and $\log n$-wise independence.

In this work we obtain an exponential efficiency improvement (even with respect to the second construction of [CRSW13]), while maintaining the best known seed length (that of their first construction).

Construction 3 (Load-Balancing Hash Family). *We construct a family of load-balancing hash functions with seed length $O(\log n\log\log n)$, which can be evaluated in $O((\log\log n)^2)$ word operations.*

In fact, this is an instantiation of the [CRSW13] construction, using the fast almost-independent hash functions of Construction 1. The efficient block computation property of Construction 1 is essential for this instantiation.

The Unit-Cost Word RAM Model. Throughout this work, we consider the *unit-cost Word RAM model* in which the elements are taken from a universe of size u, and each element can be stored in a single word of length $w = O(\log u)$ bits. Throughout this work we take the universe size to be $\text{poly}(n)$ and the word length to be $O(\log n)$ (the standard setting of parameters).

The unit cost RAM model has been the subject of much research, and is considered the standard model for analyzing the efficiency of data structures and hashing schemes; see, e.g., [DP08, Hag98, HMP01, Mil99, PP08] and the references therein. In this model, it is assumed that a certain set of operations on words come at unit cost (an abstraction for the set of instructions supported by a CPU). This set of supported operations is an important part of the model, both from a practical and from a foundational point of view. Indeed, for a given

algorithm, its running time may vary greatly depending on the set of unit-cost operations. We aim for a "minimal" model, and assume that the following (standard) operations can be executed in constant time on w-bit operands: bitwise Boolean operations, parity, left and right bit shifts by an arbitrary number of positions. We also assume constant-time addition, subtraction and multiplication over finite fields (we refer to these as *field operations* throughout). Our main results are stated assuming support for field operations over $\mathbb{GF}(2^w)$. Our constructions can also be instantiated over prime fields $\mathbb{GF}(p)$ (using addition and multiplication only), at the cost of an additional $O(\log \log n)$-overhead. See the full version for details on results over $\mathbb{GF}(p)$.

We elaborate on these two variants of supported field operations. Addition and subtraction over any finite field, and multiplication over a prime field $\mathbb{GF}(p)$, are all standard operations. Multiplication over $\mathbb{GF}(2^w)$, which requires taking the polynomial product modulo an irreducible polynomial, is used in a wide range of applications (and implementations) including error correcting codes [RS60, LRPP09, RCL$^+$13] and cryptography [oSN01, Dwo07]. Notably, modern processors provide (partial) support for multiplication over binary fields [GK12].

We avoid more complex (and non-standard) field operations such as division, inversion, or exponentiation, which are considered to be expensive operations (see, e.g., [Ram96], [And96] for two works working in the unit-cost Word RAM model where special care is taken to avoid division). We note, however, that if we allow such powerful unit-cost operations, then simpler and more efficient solutions are known. For example, if we allow unit-cost *exponentiation*, then the "powering construction" of [AGHP92] is a poly$(1/n)$-biased generator that can be evaluated in $O(1)$ time. A construction obtaining a poly$(1/n)$-biased generator with $O(\log n)$ seed length, using $O(1)$ divisions, was communicated to us by Zuckerman [Zuc].

1.1 Overview of Constructions and Techniques

Our constructions combine both algebraic as well as combinatorial techniques from the works of Naor and Naor [NN93] and Alon *et al.* [AGHP92], in addition to employing the load-balancing hash function construction of to Celis *et al.* [CRSW13]. We start by constructing (almost) $O(\log n)$-wise independent Boolean hash functions, which can be used to construct hash functions for load balancing. We then combine the two constructions ($O(\log n)$-wise independence and load balancing) to construct poly$(1/n)$-biased sequences. Thus, the $O(\log n)$-wise independent family is used both directly, and as part of the load balancing construction. In other words, our construction can be viewed as going from ε-bias (against logarithmically sparse tests) to load balancing, and back again to (full-fledged) ε-bias.

Looking further "under the hood", the starting point of our construction of $O(\log n)$-wise independence is one of the ε-bias constructions from [AGHP92]. In this respect, our construction can be viewed as transforming a (not efficient enough) ε-biased generator to a more efficient $O(\log n)$-wise independent hash

family, and back again to a (still efficient) ε-biased generator. (See the full version for an elaboration on this perspective.)

We proceed with a more detailed overview of constructions and techniques. We begin with our construction of $O(\log n)$-wise independence directly. As mentioned above, (almost) $\log n$-wise independence follows from fooling (logarithmically) sparse linear tests.

(k, ε)-Biased Generator. We wish to construct a generator G which fools k-sparse linear tests in the sense of satisfying Eq. (1.1) for vectors $\mathbf{t}$ which have at most k non-zero coordinates. For $m = O(k + \log(n/\varepsilon))$, the generator G stretches a $2m$-bit seed[1] to nm bits (alternatively, to n blocks of m output bits), where each m-bit output block can be computed very efficiently. The construction uses the field $\mathbb{GF}(2^m)$.[2] The generator's seed consists of two field elements $\beta, \gamma \in \mathbb{GF}(2^m)$, and it outputs n field elements, or nm bits in total. We index these output elements using an arbitrary subset $A \subseteq \mathbb{GF}(2^m)$ of size n. For $\alpha \in A$, the α-th output element is:

$$(G(\beta,\gamma))_\alpha \stackrel{\text{def}}{=} \gamma \cdot \sum_{i=0}^{k-1} (\alpha\beta)^i, \tag{1.2}$$

and we treat each of these field elements as an m-bit block.

Fast Bit and Block Computation. The main advantage of this construction is that the α-th block can be computed very efficiently, in $O(\log k)$ word operations (and so can each individual output bit). The efficient computation procedure uses the equality:

$$\begin{aligned}(G(\beta,\gamma))_\alpha &= \gamma \cdot \sum_{i=0}^{k-1} (\alpha\beta)^i = \gamma \cdot (1 + (\alpha\beta)) \cdot \sum_{i=0}^{\frac{k}{2}-1} ((\alpha\beta)^2)^i \\ &= \gamma \cdot (1+(\alpha\beta)) \cdot (1 + (\alpha\beta)^2) \cdot \sum_{i=0}^{\frac{k}{4}-1} ((\alpha\beta)^4)^i = \ldots = \gamma \cdot \prod_{j=0}^{\log k - 1} (1+(\alpha\beta)^{2^j})\end{aligned}$$

(w.l.o.g. we take k to be a power of 2)[3]. This final product can be computed using $O(\log k)$ operations on m-bit words: $2\log k + 1$ multiplications and $\log k$ additions.

[1] If either $k = \Omega(\log n)$ or n is polynomial in $1/\varepsilon$ then the seed-length is optimal up to a constant factor. In general, though, the optimal dependence of the seed on n is additive $\log\log n$. See the full version for details.

[2] The construction can be revised, using a transformation due to Rao [Rao07], to use instead the field $\mathbb{GF}(p)$ for some prime p. This incurs a $\log\log n$ overhead in both efficiency and seed length. See the full version for more details. As this generator is the main building block we use, all other constructions in the paper can similarly be made to work with operations over $\mathbb{GF}(p)$.

[3] We can also write $G(\beta,\gamma) = \gamma\cdot((\alpha\beta)^{k+1}-1))/(\alpha\beta-1)$ and compute it using $O(\log k)$ multiplications and one division. Here we try to avoid using divisions.

One parameter range of particular interest is a $(\log n, \text{poly}(1/n))$-biased generator. For this setting of parameters, each output block is of length $O(\log n)$, and can be computed using $O(\log \log n)$ word operations (on $(\log n)$-bit words). As discussed above, this generator immediately yields a $\text{poly}(1/n)$-almost $\log n$-wise independent Boolean hash family, where the hash functions can be evaluated in $O(\log \log n)$ time.

Proof of Pseudorandomness. We would like to prove that G is a (k, ε)-biased generator. Namely that G produces a distribution over nm-bit strings that fools k-sparse linear tests over $\mathbb{GF}(2)$. Towards this, we first show that when viewing the output of G as n field elements of $\mathbb{GF}(2^m)$, it fools any k-sparse linear test over $\mathbb{GF}(2^m)$. We then prove that *every* (k, ε)-biased generator for tests over $\mathbb{GF}(2^m)$ remains (k, ε)-biased when we interpret its output as bits. This is proven by showing that a k-sparse linear test over $\mathbb{GF}(2)$ can be "simulated" by a k-sparse linear test over $\mathbb{GF}(2^m)$.

The direct proof that G is a (k, ε)-biased generator is fairly simple. Still we describe the way we obtained the construction, as we hope that this approach may find further applications. Our starting point is an ε-biased generator due to [AGHP92] (slightly modified to generate ε-biased sequences over $\mathbb{GF}(2^m)$ rather than over $\mathbb{GF}(2)$). By itself, the generator is not efficient enough (as most output elements require $\Omega(\log n)$ operations to evaluate). To improve construction's efficiency, we use a reduction due to [NN93] from ε-bias to (k, ε)-bias. This reduction is used in [NN93] to *reduce the seed length*.[4] Our goal is quite different: we will use the reduction to *improve efficiency*, obtaining (k, ε)-biased sequences that require $\log k$ operations (rather than $\log n$). A priori, there is no reason to expect the reduction to have this consequence, but a careful instantiation of the reduction does work.

Almost k-Wise Independence and Load Balancing. The $(\log n, \text{poly}(1/n))$-biased generator obtained above directly implies a $\text{poly}(1/n)$-almost $\log n$-wise independent Boolean hash family, where functions in the family can be evaluated in $O(\log \log n)$ word operations. Moreover, because of the efficient block computation property, for every $t \in \{1, 2, 3, \ldots \log n\}$, we also get a $\text{poly}(1/n)$-almost $(\log n/t)$-wise independent hash family from $[n]$ to $\{0,1\}^t$, where the functions can be evaluated in $O(\log \log n)$ word operations (we emphasize that this is the cost to obtain *all* of the output bits simultaneously). For every such t, a hash function h is described by two $GF(2^m)$ elements β and γ and $h_{\beta,\gamma}(i)$ is defined to be the t-long suffix of $G(\beta, \gamma))_{\alpha_i}$, where α_i is the i'th field element in A.

Such almost k-wise independent functions are used in [CRSW13] to construct load-balancing hash functions. The [CRSW13] construction uses $\log \log n$ such hash functions (each of these hash functions is evaluated once). The w-th function, for $w \in [\log \log n]$, is taken from a family that is (almost) $O(\log n/2^w)$-wise independent with 2^w-bit outputs. We instantiate their construction with our new

[4] More specifically, the dependence on n in the seed of (k, ε)-biased sequences is better than the dependence for ε-bias sequences.

hash families, improving the evaluation time from $\tilde{O}(\log n)$ to $O((\log\log n)^2)$ word operations. The seed length remains $O(\log n \cdot \log\log n)$.

(Full-Fledged) ε-Biased Generator. Having used a $(\log n, \text{poly}(1/n))$-biased generator to obtain fast load-balancing, we now use fast load-balancing (together with the $(\log n, \text{poly}(1/n))$-biased generator again) to obtain a fast full-fledged ε-biased generator (for *all* tests, i.e. without any sparsity restriction). This part of our construction is inspired by the combinatorial techniques of [NN93]. Special care is needed to simultaneously preserve both the small error as well as the efficient computation. To handle this we introduce a stronger notion of load balancing we call "granular" load balancing, which may be of interest elsewhere. See the full version for details.

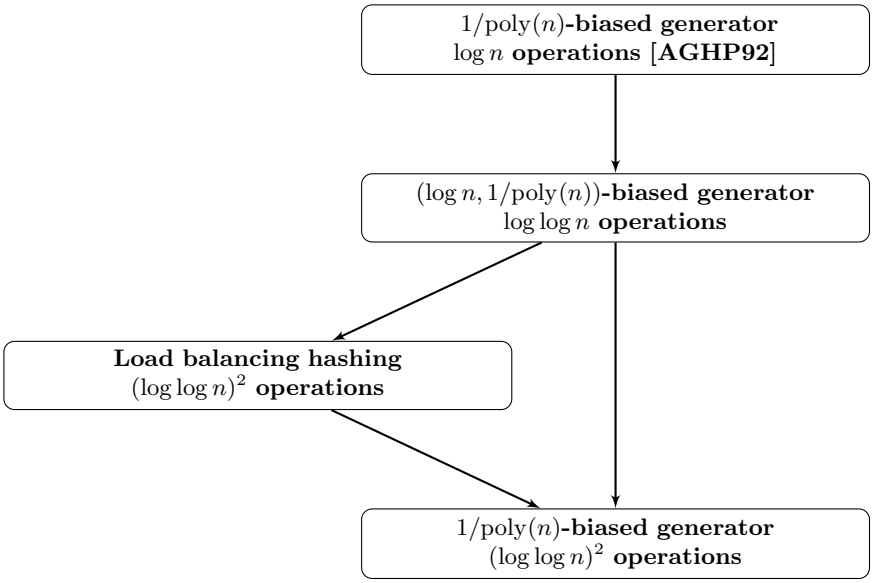

Fig. 1. Roadmap for the construction of the fast ε-biased generator

Acknowledgments. We would like to thank Udi Wieder and Moni Naor for useful discussions.

References

ADM+99. Alon, N., Dietzfelbinger, M., Miltersen, P.B., Petrank, E., Tardos, G.: Linear hash functions. J. ACM 46(5), 667–683 (1999)

AGHP92. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple construction of almost k-wise independent random variables. Random Struct. Algorithms 3(3), 289–304 (1992)

AIK+90. Ajtai, M., Iwaniec, H., Komlós, J., Pintz, J., Szemerédi, E.: Construction of a thin set with small Fourier coefficients. Bulletin of the London Mathematical Society 22, 583–590 (1990)

AIK06. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. SIAM J. Comput. 36(4), 845–888 (2006)

AM95. Alon, N., Mansour, Y.: Epsilon-discrepancy sets and their application for interpolation of sparse polynomials. Inf. Process. Lett. 54(6), 337–342 (1995)

AMN98. Azar, Y., Motwani, R., Naor, J.: Approximating probability distributions using small sample spaces. Combinatorica 18(2), 151–171 (1998)

And96. Andersson, A.: Faster deterministic sorting and searching in linear space. In: FOCS, pp. 135–144 (1996)

AR94. Alon, N., Roichman, Y.: Random Cayley graphs and expanders. Random Struct. Algorithms 5(2), 271–285 (1994)

BATS13. Ben-Aroya, A., Ta-Shma, A.: Constructing small-bias sets from algebraic-geometric codes. Theory of Computing 9, 253–272 (2013)

BFLS91. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: STOC, pp. 21–31 (1991)

BSGH+06. Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Robust PCPs of proximity, shorter PCPs, and applications to coding. SIAM J. Comput. 36(4), 889–974 (2006)

BV10. Bogdanov, A., Viola, E.: Pseudorandom bits for polynomials. SIAM J. Comput. 39(6), 2464–2486 (2010)

CM01. Cryan, M., Miltersen, P.B.: On pseudorandom generators in NC. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 272–284. Springer, Heidelberg (2001)

CRSW13. Elisa Celis, L.: Omer Reingold, Gil Segev, and Udi Wieder. Balls and bins: Smaller hash families and faster evaluation. SIAM J. Comput. 42(3), 1030–1050 (2013)

CW79. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. 18(2), 143–154 (1979)

DP08. Dietzfelbinger, M., Pagh, R.: Succinct data structures for retrieval and approximate membership (extended abstract). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 385–396. Springer, Heidelberg (2008)

Dwo07. Dworkin, M.: Recommendations for Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D (2007)

EGL+98. Even, G., Goldreich, O., Luby, M., Nisan, N., Velickovic, B.: Efficient approximation of product distributions. Random Struct. Algorithms 13(1), 1–16 (1998)

FGL+96. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. J. ACM 43(2), 268–292 (1996)

GK12. Gueron, S., Kounavis, M.E.: Intel® carry-less multiplication instruction and its usage for computing the GCM mode, rev 2.01 (2012)

GMR+12. Gopalan, P., Meka, R., Reingold, O., Trevisan, L., Vadhan, S.P.: Better pseudorandom generators from milder pseudorandom restrictions. In: FOCS, pp. 120–129 (2012)

GV04. Gutfreund, D., Viola, E.: Fooling parity tests with parity gates. In: Jansen, K., Khanna, S., Rolim, J.D.P., Ron, D. (eds.) RANDOM and APPROX 2004. LNCS, vol. 3122, pp. 381–392. Springer, Heidelberg (2004)

Hag98. Hagerup, T.: Sorting and searching on the word RAM. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 366–398. Springer, Heidelberg (1998)

Hea08. Healy, A.: Randomness-efficient sampling within nc^1. Computational Complexity 17(1), 3–37 (2008)
HMP01. Hagerup, T., Miltersen, P.B., Pagh, R.: Deterministic dictionaries. J. Algorithms 41(1), 69–85 (2001)
HV06. Healy, A., Viola, E.: Constant-depth circuits for arithmetic in finite fields of characteristic two. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 672–683. Springer, Heidelberg (2006)
Lov09. Lovett, S.: Unconditional pseudorandom generators for low degree polynomials. Theory of Computing 5(1), 69–82 (2009)
LRPP09. Lacan, J., Roca, V., Peltotalo, J., Peltotalo, S.: Reed-Solomon Forward Error Correction (FEC) Schemes. RFC 5510 (Proposed Standard) (April 2009)
LW05. Luby, M., Wigderson, A.: Pairwise independence and derandomization. Foundations and Trends in Theoretical Computer Science 1(4) (2005)
Mil99. Miltersen, P.B.: Cell probe complexity - a survey. In: 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 1999. Advances in Data Structures Workshop (1999)
MST06. Mossel, E., Shpilka, A., Trevisan, L.: On epsilon-biased generators in NC^0. Random Struct. Algorithms 29(1), 56–81 (2006)
MW04. Meshulam, R., Wigderson, A.: Expanders in group algebras. Combinatorica 24(4), 659–680 (2004)
Nao92. Naor, M.: Constructing Ramsey graphs from small probability spaces. IMB Research Report RJ(8810) (1992)
NN93. Naor, J., Naor, M.: Small-bias probability spaces: Efficient constructions and applications. SIAM J. Comput. 22(4), 838–856 (1993)
oSN01. National Institute of Standards and Technology (NIST). Federal information processing standards publication (FIPS 197). Advanced Encryption Standard (AES) (2001)
PP08. Pagh, A., Pagh, R.: Uniform hashing in constant time and optimal space. SIAM J. Comput. 38(1), 85–96 (2008)
PPR07. Pagh, A., Pagh, R., Ruzic, M.: Linear probing with constant independence. In: STOC, pp. 318–327 (2007)
Ram96. Raman, R.: Priority queues: Small, monotone and trans-dichotomous. In: Díaz, J. (ed.) ESA 1996. LNCS, vol. 1136, pp. 121–137. Springer, Heidelberg (1996)
Rao07. Rao, A.: An exposition of Bourgain's 2-source extractor. Electronic Colloquium on Computational Complexity (ECCC) 14(034) (2007)
Raz05. Raz, R.: Extractors with weak random seeds. In: STOC, pp. 11–20 (2005)
RCL+13. Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., Matsuzono, K.: Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME. RFC 6865 (Proposed Standard) (February 2013)
RRW14. Reingold, O., Rothblum, R.D., Wieder, U.: Pseudorandom graphs in data structures (manuscript, 2014)
RS60. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8(2), 300–304 (1960)
RSW93. Razborov, A.A., Szemerédi, E., Wigderson, A.: Constructing small sets that are uniform in arithmetic progressions. Combinatorics, Probability & Computing 2, 513–518 (1993)
RV14. Reingold, O., Vardi, S.: Tighter bounds for local computations (manuscript, 2014)
Shp09. Shpilka, A.: Constructions of low-degree and error-correcting epsilon-biased generators. Computational Complexity 18(4), 495–525 (2009)

Sie04. Siegel, A.: On universal classes of extremely random constant-time hash functions. SIAM J. Comput. 33(3), 505–543 (2004)
Tho13. Thorup, M.: Simple tabulation, fast expanders, double tabulation, and high independence. In: FOCS (2013)
Vio09. Viola, E.: The sum of d small-bias generators fools polynomials of degree d. Computational Complexity 18(2), 209–217 (2009)
WC81. Wegman, M.N., Carter, L.: New hash functions and their use in authentication and set equality. J. Comput. Syst. Sci. 22(3), 265–279 (1981)
Zuc. Zuckerman, D.: Personal Communication