A Note on Randomness Extraction

This version of the document is dated 2020-08-17.

Peter Occil

Randomness extraction (also known as *unbiasing*, *debiasing*, *deskewing*, *whitening*, or *entropy extraction*) is a set of techniques for generating unbiased random bits from biased sources. This note covers some useful extraction techniques.

0.1 In Information Security

In information security, randomness extraction serves to generate a seed, password, encryption key, or other secret value from hard-to-predict sources of noise.

Randomness extraction for information security is discussed in NIST SP 800-90B sec. 3.1.5.1, and RFC 4086 sec. 4.2 and 5.2. Popular choices of such extractors include keyed cryptographic hash functions (see, e.g., (Cliff et al., 2009)**(1)**. In information security applications, extractors other than keyed cryptographic hash functions should not be used by themselves in randomness extraction.

Some papers also refer to two-source extractors and resilient functions (especially the works by E. Chattopadhyay and D. Zuckerman), but there are few if any real implementations of these extraction techniques.

> **Example:** The Cliff reference reviewed the use of HMAC (hash-based message authentication code) algorithms, and implies that one way to generate a seed is as follows:
>
> 1. Gather data with at least 512 bits of entropy.
> 2. Run HMAC-SHA-512 with that data to generate a 512-bit HMAC.
> 3. Take the first 170 (or fewer) bits as the seed (512 divided by 3, rounded down).

0.2 Outside of Information Security

Outside of information security, randomness extraction serves the purpose of recycling random numbers, to reduce calls to a pseudorandom number generator (PRNG), for example. This is in addition to the purpose of generating a seed for a PRNG.

Perhaps the most familiar example of randomness extraction is the one by von Neumann (1951)**(2)**:

1. Flip a coin twice (whose bias is unknown).
2. If the coin lands heads-tails or tails-heads, return heads or tails, respectively. Otherwise, go to step 1.

An algorithm from (Morina et al. 2019)**(3)** extends this to loaded dice. Based on personal communication by K. Łatuszyński, perhaps this works for any distribution of random numbers, not just loaded dice, as the key "is to find two non overlapping events of the same probability" via "symmetric events {X_1 < X_2} and {X_2 < X_1} that have the same probability".

1. Throw a die twice (whose bias is unknown), call the results X and Y, respectively.
2. If X is less than Y, return 1. If X is greater than Y, return 0. Otherwise, go to step 1.

Pae (2005)**[4]** characterizes *extracting functions*. Informally, an *extracting function* is a function that maps a fixed number of bits to a variable number of bits such that, whenever the input has a given number of ones, every output string of a given length is as likely to occur as every other output string of that length, regardless of the input coin's bias. Among others, von Neumann's extractor and the one by Peres (1992)**[5]** are extracting functions.

A streaming algorithm, which builds something like an "extractor tree", is another example of a randomness extractor <<Zhou, H. and Bruck, J., "**Streaming algorithms for optimal generation of random bits**", arXiv:1209.0730 [cs.IT], 2012.>>.

I maintain **source code of this extractor**, which also includes additional notes on randomness extraction.

Pae's "entropy-preserving" binarization (Pae 2020)**[6]**, given below, is meant to be used in other extractor algorithms such as the ones mentioned above. It assumes the number of possible values n is known. However, it is obviously not efficient if n is a large number.

1. Let f be a number in the interval 0, n) that was previously randomly generated. If f is greater than 0, output a 1.
2. If f is less than $n - 1$, output a 0 x times, where x is $(n - 1) - f$.

Finally, different kinds of random numbers should not be mixed in the same extractor stream. For example, if one source outputs random 6-sided die results, another source outputs random sums of rolling 2 six-sided dice, and a third source outputs coin flips with a bias of 0.75, there should be three extractor streams (for instance, three extractor trees that implement the Zhou and Bruck algorithm).

The lower bound on the average number of coin flips needed to turn a biased coin into an unbiased coin is as follows (and is a special case of the *entropy bound*; see, e.g., (Pae 2005)[**[7]**, (Peres 1992)**[8]**):

$$\ln(2) / ((\lambda - 1) * \ln(1 - \lambda) - \lambda * \ln(\lambda)),$$

where λ is the bias of the input coin. According to this formula, a growing number of coin flips is needed if the input coin is strongly biased towards heads or tails.

1 Notes

[1] Cliff, Y., Boyd, C., Gonzalez Nieto, J. "How to Extract and Expand Randomness: A Summary and Explanation of Existing Results", 2009.

[2] von Neumann, J., "Various techniques used in connection with random digits", 1951.

[3] Morina, G., Łatuszyński, K., et al., "From the Bernoulli Factory to a Dice Enterprise via Perfect Sampling of Markov Chains", arXiv:1912.09229v1 [math.PR], 2019.

[4] Pae, S., "Random number generation using a biased source", dissertation, University of Illinois at Urbana-Champaign, 2005.

[5] Peres, Y., "Iterating von Neumann's procedure for extracting random bits", Annals of Statistics 1992,20,1, p. 590-597.

[6] S. Pae, "Binarization Trees and Random Number Generation", arXiv:1602.06058v2 [cs.DS].

[7] Pae, S., "Random number generation using a biased source", dissertation, University of Illinois at Urbana-Champaign, 2005.

[8] Peres, Y., "Iterating von Neumann's procedure for extracting random bits", Annals of Statistics 1992,20,1, p. 590-597.

2 License