Extracting quasi-uniform bits from a coin of known bias

Vinícius G. Pereira de Sá[1], **Marcio N. de Miranda**[2], **Daniel S. Menasché**[1]

[1]Departamento de Ciência da Computação – Univ. Federal do Rio de Janeiro (UFRJ)
Caixa Postal 68530 – CEP 21941-590 – Rio de Janeiro – RJ – Brazil

[2]Departmento de Matemática – Univ. Federal Rural do Rio de Janeiro (UFRRJ)

{sadoc,vigusmao}@dcc.ufrj.br, marcionmiranda@ufrrj.br

Abstract. *The problem of producing uniform bits out of a non-uniform source of randomness has utmost theoretical and practical relevance. A common metaphor for one such source is that of a possibly biased coin which yields heads with probability* $0 < p < 1$*. The state-of-the-art algorithms to produce uniform bits in such setting do not employ any knowledge of* p*, an information which is readily available—or can be inferred—in many cases. Moreover, strict uniformity is not always a requirement, and in many applications it may be perfectly acceptable that the relative frequencies of '0's and '1's belong to* $[0.5-\epsilon, 0.5+\epsilon]$*, for some* $\epsilon \geq 0$*. We introduce an algorithm which benefits from the knowledge of* p *to produce quasi-uniform bits with high bit-to-flip ratios, unconstrained by the well-known upper bound—for strictly uniform distribution—given by Shannon's entropy function.*

Resumo. *O problema de se produzir bits uniformes a partir de fontes não-uniformes de aleatoriedade tem imensa relevância teórica e prática. Uma metáfora comum é o lançamento de uma moeda possivelmente viciada que produz cara com probabilidade* $0 < p < 1$*. No atual estado da arte, os algoritmos que produzem bits uniformes nesse cenário não utilizam qualquer conhecimento sobre o viés* p *da moeda, uma informação que pode, no entanto, estar disponível—ou ser inferida—em muitos casos. Além disso, uma uniformidade estrita não é sempre realmente necessária, sendo perfeitamente aceitável em muitas aplicações que as frequências relativas de '0's e '1's pertençam a* $[0.5 - \epsilon, 0.5 + \epsilon]$*, para algum* $\epsilon \geq 0$*. Apresentamos um algoritmo que se beneficia do conhecimento de* p *para produzir bits quase-uniformes com altas taxas de bits por laçamento da moeda, não sujeitas ao conhecido limite superior—para distribuição uniforme—dado pela função da entropia de Shannon.*

1. Introduction

The ability to obtain uniform random numbers plays a fundamental role in a number of applications. Such applications include computer simulation, gambling, statistical sampling, network protocols, cryptography, hashing and randomized algorithms, to name but a few. Not surprisingly, generating uniform bits out of a non-uniform source of randomness *efficiently* is a well-known engineering problem for which plenty of research has been devoted [Kroese et al. 2011]. It is often regarded as an optimization problem whose goal is to minimize the expected number f of flips of a "biased coin" for each uniform bit

that is produced. Or, equivalently, to maximize $1/f$, the expected *bit-to-flip ratio* (BTFR) of the process.

Given a coin that produces heads ($\mathcal{H}$) with probability $0 < p < 1$ and tails ($\mathcal{T}$) with probability $1 - p$, the celebrated method proposed by von Neumann in the early 1950s [von Neumann 1951] achieves a BTFR of $p(1 - p)$. The idea is to toss the coin twice in a row and output a '1' if the results were $\mathcal{HT}$, a '0' if they were $\mathcal{TH}$, starting over otherwise with two fresh tosses.

After von Neumann's seminal paper, there followed several algorithms addressing the problem of obtaining uniform randomness from non-uniform sources. For example, [Hoeffding and Simons 1970] exhibits and analyzes procedures for converting a sequence of i.i.d. Bernoulli variables with unknown mean p into a Bernoulli variable with mean 0.5; [Elias 1972] generalizes the results of Hoeffding and Simons, converting input sequences of symbols generated by a stationary random process into sequences of independent and equiprobable output symbols; [Stout and Warren 1984] and [Stout 1985] present tree-based algorithms which take as input a sequence of i.i.d. discrete random variables and output a uniform random variable with $n > 0$ possible outcomes; and, more recently, in [Naor and Reingold 2004] and [Zhou and Bruck 2012], different schemes for generating a predefined number of random bits out of biased coins are discussed.

A series of direct improvements over the original von Neumann's method has also been proposed. For instance, the method presented in [Peres 1992] achieves the BTFR given by the recursive relation

$$B(p) := p(1-p) + \frac{p^2 + (1-p)^2}{2} \cdot B\left(\frac{p^2}{p^2 + (1-p)^2}\right).$$

Even though no closed formula is known for $B(p)$, it is easy to check that it significantly outperforms von Neumann's method for all $p > 0$.

Exploiting known biases. All those methods tackle the bit generation problem without requiring any knowledge of p, that is, they work for whatever undisclosed probability $0 < p < 1$ that our metaphorical coin lands heads up. However, if one *knows* the bias p of the coin, one might be able exploit such knowledge to improve the BTFR. Indeed, if $p = 0.5$ (a fair coin), the BTFR obtained by the original von Neumann's method and by Peres's method would be, respectively, 0.25 and $0.333\ldots$, whereas a BTFR of 1 could be obtained trivially (by a simple $\mathcal{H} \rightarrow$'0', $\mathcal{T} \rightarrow$'1' mapping).

For a less obvious example, consider the case where $p = 0.2$. In this case, the strategy of tossing the coin *four* times in a row, producing a '0' if the sequence of results belongs to $\{\mathcal{TTTT}, \mathcal{THTH}, \mathcal{HHHT}, \mathcal{HHTT}, \mathcal{HTHT}, \mathcal{HTHH}\}$ and '1' if it belongs to $\{\mathcal{THHT}, \mathcal{THHH}, \mathcal{TTTH}, \mathcal{TTHH}, \mathcal{TTHT}, \mathcal{THTT}, \mathcal{HHTH}, \mathcal{HTTH}, \mathcal{HTTT}\}$ is perfectly fair (each new bit is produced with probability 0.4992 after a series of four flips), yielding a BTFR of 0.2496. In contrast, the BTFR obtained by von Neumann's method for this value of p is $p(1 - p) = 0.16$, and the BTFR of Peres's method is $B(p) \approx 0.179$.

Quasi-uniform bits. In many applications, a slight non-uniformity among the generated bits might be acceptable, so a typical quality-versus-time tradeoff ensues: is it worth relaxing the uniformity requirement to increase the BTFR? The general (relaxed) bit-generation problem can be formulated as follows.

INPUT: a sequence of independent Bernoulli trials with success probability $0 < p < 1$ used as source of randomness (a "biased coin"), and some *tolerance* $\epsilon > 0$;

OUTPUT: a "random" sequence of bits, so that the relative frequencies of '0's and '1's belong, on average, to $[1 - \epsilon, 1 + \epsilon]$.

Infinitely many algorithms—with different BTFR's—can be devised to convert the input into the desired output. When $\epsilon = 0$ (uniform distribution of '0's and '1's), the algorithm presented in [Mitzenmacher 2002] achieves a BTFR that equals the known upper bound given by the entropy function [Shannon 2001]

$$H(p) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

when the number of flips goes to infinity, while not demanding any knowledge of p. In the relaxed setting, however, the entropy is not any longer an upper bound. Indeed, given a certain tolerance ϵ strictly greater than zero, if $p = 0.5 - \alpha$ for some $\alpha < \epsilon$, then $H(p) < 1$, whereas a BTFR of 1 can be obtained trivially ($\mathcal{H} \rightarrow$ '0', $\mathcal{T} \rightarrow$ '1'). The following (meta-)problem thus arises: given p and ϵ as discussed, what is the best *algorithm* (i.e., the conversion strategy with the greatest BTFR) to translate the input sequence into the desired quasi-uniform bit sequence?

Our contribution. In this paper, we describe a simple method to answer the aforementioned problem, exploiting the knowledge of p to obtain conversion strategies with higher BTFR's (as compared to those produced by uniformity-oriented algorithms) for the price of a slight non-uniformity—bounded by ϵ—in the distribution of the generated bits.

After introducing our algorithm (Section 2) and presenting some preliminary computational results (Section 3), we conclude the paper (Section 4) by discussing some future possibilities.

2. An iterative partitioning algorithm

Suppose we flip k times in a row a coin which yields heads with probability $0 < p < 1$. There are 2^k possible outcomes, in the form of sequences of symbols $\mathcal{H}$ and $\mathcal{T}$. Let $\mathcal{P}(k)$ be the set of all such sequences. Among them, there are $\binom{k}{h}$ sequences containing exactly h symbols $\mathcal{H}$ (and $k - h$ symbols $\mathcal{T}$), for $0 \leq h \leq k$, and each one of them occurs with probability $q(h, k) := p^h (1 - p)^{k-h}$.

Let S be a subset of $\mathcal{P}(k)$. We define $q(S)$ as the probability that one of the sequences in S happens to be the outcome of a series of k flips of our coin. Since all sequences are mutually exclusive, such probability corresponds to the sum of the individual probabilities of occurrence of the sequences in S.

Now let $\epsilon > 0$, and suppose there are two disjoint subsets S_0 and S_1 of $\mathcal{P}(k)$ such that

$$f(S_0) := \frac{q(S_0)}{q(S_0) + q(S_1)} \in [0.5 - \epsilon, 0.5 + \epsilon]. \tag{1}$$

Algorithm 1 *find_best_strategy* (p, ϵ, k_{max})

input: the probability $0 < p < 1$ that a coin lands heads up,
a tolerance $\epsilon \geq 0$,
and a stop parameter k_{max}

output: a pair (S_0, S_1), where S_0 is the set of coin sequences that should produce a bit '0'
and S_1 is the set of coin sequences that should produce a bit '1'

1. *best_btfr* $\leftarrow 0$
2. *best_strategy* $\leftarrow$ None
3. **for each** $k \in \{1, 2, \ldots, k_{max}\}$ **do**
4. $P(k) \leftarrow \{(\mathcal{H}|\mathcal{T})^k\}$ // the set of all possible sequences of k symbols $\mathcal{H}$ or $\mathcal{T}$
5. **for all** $S_0, S_1 \subset \mathcal{P}(k)$ such that $S_0 \cap S_1 = \emptyset$ **do**
6. $h_0 \leftarrow$ number of symbols $\mathcal{H}$ in S_0
7. $q(S_0) \leftarrow p^{h_0}(1-p)^{k-h_0}$
8. $h_1 \leftarrow$ number of symbols $\mathcal{H}$ in S_1
9. $q(S_1) \leftarrow p^{h_1}(1-p)^{k-h_1}$
10. $f(S_0) \leftarrow \frac{q(S_0)}{q(S_0)+q(S_1)}$
11. **if** $f(S_0) \in [0.5 - \epsilon, 0.5 + \epsilon]$ **then**
12. $btfr \leftarrow q(S_0) + q(S_1)$
13. **if** *btfr* $>$ *best_btfr* **then**
14. *best_btfr* $\leftarrow$ *btfr*
15. *best_strategy* $\leftarrow (S_0, S_1)$
16. **return** *best_strategy*

In this case, we can make it so that an outcome in S_0 gives rise to a bit '0', and an outcome in S_1 gives rise to a bit '1'. Note that $f(S_0)$ is the expected ratio of bits '0' over all bits that are generated.

The probability of producing *any* bit after a series of k flips is $q(S_0) + q(S_1)$ (they are disjoint), so that the BTFR of such a strategy would be $[q(S_0) + q(S_1)]/k$.

The input of our algorithm—which is given in pseudocode as Algorithm 1—is the probability p of obtaining $\mathcal{H}$, the tolerance $\epsilon \geq 0$, and a parameter k_{max} that indicates the maximum length of series of flips that must be considered. It proceeds as follows.

For $k = 1, 2, \ldots, k_{max}$, the algorithm determines the strategy with the highest BTFR considering only series of k consecutive flips of the coin. So, for each value of k, it starts by generating the set $\mathcal{P}(k)$. Now, for each pair (S_0, S_1) of disjoint subsets of $\mathcal{P}(k)$, it checks whether the inequality (1) is satisfied, keeping track of the pair that yields the greatest BTFR, which is returned at the end.

The time complexity of the proposed algorithm is clearly exponential in k_{max}. However, for very small values of k_{max}, and even for narrow tolerances ϵ, it is possible to obtain rather good results, as our computational results show in Section 3.

3. Preliminary results

In order to illustrate the possibilities of the proposed method, we present some of the computational results[1] that we produced for different values of p and ϵ, all of them using $k_{max} = 4$. It is possible to compare the obtained BTFR's with those of von Neumann's method and Peres's method for the same values of p. The relative frequencies $f(S_0)$ of bits '0' that are produced are also indicated.

$p = 0.2$
BTFR(von Neumann) = 0.16, BTFR(Peres) ≈ 0.1794
- $\epsilon = 0$
  - $S_0 = \{\mathcal{TTTT}, \mathcal{THTH}, \mathcal{HHHT}, \mathcal{HHTT}, \mathcal{HTHT}, \mathcal{HTHH}\}$
  - $S_1 = \{\mathcal{THHT}, \mathcal{THHH}, \mathcal{TTTH}, \mathcal{TTHH}, \mathcal{TTHT}, \mathcal{THTT}, \mathcal{HHTH}, \mathcal{HTTH}, \mathcal{HTTT}\}$
  - $f(S_0) = 0.5$
  - BTFR = 0.2496
- $\epsilon = 0.05$
  - $S_0 = \{\mathcal{TTT}\}$
  - $S_1 = \{\mathcal{THT}, \mathcal{THH}, \mathcal{TTH}, \mathcal{HHH}, \mathcal{HHT}, \mathcal{HTT}, \mathcal{HTH}\}$
  - $f(S_0) = 0.512$
  - BTFR = $0.333\ldots$
- $\epsilon = 0.15$
  - $S_0 = \{\mathcal{TT}\}$
  - $S_1 = \{\mathcal{HH}, \mathcal{HT}, \mathcal{TH}\}$
  - $f(S_0) = 0.64$
  - BTFR = 0.5

$p = 0.25$
BTFR(von Neumann) = 0.1875, BTFR(Peres) ≈ 0.2172
- $\epsilon = 0$
  - $S_0 = \{\mathcal{TTT}, \mathcal{HTH}\}$
  - $S_1 = \{\mathcal{THT}, \mathcal{THH}, \mathcal{TTH}, \mathcal{HTT}\}$
  - $f(S_0) = 0.5$
  - BTFR = 0.3125
- $\epsilon = 0.05$
  - $S_0 = \{\mathcal{TTT}, \mathcal{HHT}\}$
  - $S_1 = \{\mathcal{THT}, \mathcal{THH}, \mathcal{TTH}, \mathcal{HHH}, \mathcal{HTT}, \mathcal{HTH}\}$
  - $f(S_0) = 0.46875$
  - BTFR = $0.333\ldots$
- $\epsilon = 0.07$
  - $S_0 = \{\mathcal{TT}\}$
  - $S_1 = \{\mathcal{HH}, \mathcal{HT}, \mathcal{TH}\}$
  - $f(S_0) = 0.5625$
  - BTFR = 0.5

[1] The code, written in Python 3, can be found in `https://www.dropbox.com/s/ts6hs063wxoe5rl/btfr.py`.

$p = 0.3$
BTFR(von Neumann) = 0.21, BTFR(Peres) ≈ 0.2514
- $\epsilon = 0$
  - $S_0 = \{\mathcal{TH}\}$
  - $S_1 = \{\mathcal{HT}\}$
  - $f(S_0) = 0.5$
  - BTFR = 0.21
- $\epsilon = 0.05$
  - $S_0 = \{\mathcal{TT}\}$
  - $S_1 = \{\mathcal{HH}, \mathcal{HT}, \mathcal{TH}\}$
  - $f(S_0) = 0.49$
  - BTFR = 0.5

$p = 0.4$
BTFR(von Neumann) = 0.24, BTFR(Peres) ≈ 0.3067
- $\epsilon = 0$
  - $S_0 = \{\mathcal{TH}\}$
  - $S_1 = \{\mathcal{HT}\}$
  - $f(S_0) = 0.5$
  - BTFR = 0.24
- $\epsilon = 0.05$
  - $S_0 = \{\mathcal{HH}, \mathcal{TT}\}$
  - $S_1 = \{\mathcal{HT}, \mathcal{TH}\}$
  - $f(S_0) = 0.52$
  - BTFR = 0.5

The following, last example is particularly interesting, since it shows that it is possible to obtain (not in a trivial way, as mentioned in Section 1) a BTFR that is greater than the entropy.

$p = 0.11$
BTFR(von Neumann) = 0.0979, BTFR(Peres) ≈ 0.1039
- $\epsilon = 0.3$
  - $S_0 = \{\mathcal{TT}\}$
  - $S_1 = \{\mathcal{HH}, \mathcal{HT}, \mathcal{TH}\}$
  - $f(S_0) = 0.7921$
  - BTFR = 0.5 (for an entropy of 0.49991596)

An interesting enhancement to our algorithm can be obtained by inspecting the sets S_0, S_1 of coin sequences that generate each bit. If one of the sets contains *all* possible sequences with a same prefix, than the remaining coin flips (after the prefix is obtained) are not even necessary, and the BTFR increases accordingly. Take, for instance, the case where p is the only real solution of

$$p^3 + p^2(1-p) - (1-p)^3 = 0,$$

namely $p \approx 0.43016$. In this case, tossing the coin three times in a row, producing a '0' if the resulting sequence belongs to $\{\mathcal{HHH}, \mathcal{HHT}, \mathcal{HTH}, \mathcal{HTT}\}$ and '1' if it belongs to $\{\mathcal{THH}, \mathcal{THT}, \mathcal{TTT}\}$ is acceptable even for $\epsilon = 0$, and the resulting BTFR is approximately 0.286. Note, however, that all possible sequences starting with $\mathcal{H}$ produce a '0', and all possible sequences starting with $\mathcal{TH}$ produce a '1'. A fair strategy would therefore be as follows. Flip the coin. If the result is $\mathcal{H}$, output a '0'; otherwise, flip it a second time. If the second result is $\mathcal{H}$, output a '1'; otherwise, flip it a third time. If the result is again a $\mathcal{T}$ (the third in a row), output a '1'; otherwise, restart the process. With some tedious manipulations, the BTFR can be shown to be as good as

$$\frac{-p^3 + 2p^2 - p + 1}{p^2 - 3p + 3} \approx 0.454.$$

In contrast, the BTFR obtained by von Neumann's method for this value of p is $p(1-p) \approx 0.245$, and the BTFR of Peres's method is $B(p) \approx 0.318$.

4. Conclusion and open problems

We have shown how one can possibly explore the knowledge of the bias p of a coin to devise bit-extraction strategies with high bit-to-flip-ratios when a slight non-uniformity—parameterized by some $\epsilon \geq 0$—in the distribution of the generated bits is acceptable. Each obtained strategy is so-to-say "customized" for a given (p, ϵ) pair, as it does not guarantee either the intended distribution or a good BTFR for other instances.

Concealed biased coins. A similar approach is likely to obtain good results in the *concealed biased coin* model introduced in [Pereira de Sá and de Figueiredo 2014]. In such a setting, one cannot actually know the result of each coin flip. Instead, it is only possible to infer whether the two latest flips were the same. Even though such model was first described only theoretically, it finds straightforward applications in situations where a synchronous verification of a two-state machine is performed—e.g., the value of some company's shares in the stock market being above or below a certain threshold—and an alarm is triggered whenever the latest state of the machine differs from the previous.

The best published algorithm for producing uniform bits in the concealed biased coin model generates each bit, on average, for the price of an extra flip as compared to von Neumann's method for the classic setting, namely in expected $[p(1-p)]^{-1} + 1$ flips. While that is already a significant improvement over many natural, intuitive strategies for the concealed setting, its BTFR may be further improved by a method analogous to the one presented in Section 2. We leave that as an open problem.

Markov decision processes. The problem of determining the optimal strategy (sequence-to-bit assignment) for sequences of k flips when the bias p of the coin is known can also possibly be framed as a Markov decision process. The states correspond to subsequences of $1, 2, \ldots, k$ symbols $\mathcal{H}$ or $\mathcal{T}$, indicating the latest results that were obtained, and bits are generated after certain state transitions. A simple integer linear program can thus be solved to maximize the BTFR, making sure that the relative frequency of either bit belongs to the intended, relaxed interval. The obvious advantage of such approach is

that the steady state solution of a Markov chain can be computed quite instantly. We also leave that as subject of future research.

References

Elias, P. (1972). The efficient construction of an unbiased random sequence. *The Annals of Mathematical Statistics*, 43(3):865–870.

Hoeffding, W. and Simons, G. (1970). Unbiased coin tossing with a biased coin. *The annals of mathematical statistics*, 41:341–352.

Kroese, D. P., Taimre, T., and Botev, Z. I. (2011). *Handbook of Monte Carlo Methods*. Wiley Series in Probability and Statistics. John Wiley and Sons.

Mitzenmacher, M. (2002). Unbiasing random bits. *Dr. Dobbs Journal*, 335:101–104.

Naor, M. and Reingold, O. (2004). Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 51(2):231–262.

Pereira de Sá, V. G. and de Figueiredo, C. M. H. (2014). Blind-friendly von Neumann's heads or tails. *The American Mathematical Monthly*, to appear.

Peres, Y. (1992). Iterating von neumann's procedure for extracting random bits. *The Annals of Statistics*, 20(1):590–597.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

Stout, Q. F. (1985). Tree-based graph algorithms for some parallel computers. In *ICPP*, pages 727–730.

Stout, Q. F. and Warren, B. (1984). Tree algorithms for unbiased coin tossing with a biased coin. *The Annals of Probability*, pages 212–222.

von Neumann, J. (1951). Various techniques used in connection with random digits. *J. Res. National Bureau of Standards*, 12:36–38.

Zhou, H. and Bruck, J. (2012). Efficient generation of random bits from finite state markov chains. *Information Theory, IEEE Transactions on*, 58(4):2490–2506.