

Interval Algorithms for Coin Flipping

Sung-il Pae,

Hongik University, Seoul, Korea

Summary

We discuss two refinements of Han-Hoshi interval algorithm for random number generation in the context of coin flipping. The ideas behind the refinements suggest a partial answer to the question that the original work of Han and Hoshi left: how to improve the interval subdivision process by rearranging the order of subintervals to obtain a better average cost for random number generation. Experiment results are presented to demonstrate that our refinements perform better than the original interval algorithm.

Key words:

Random number generation, coin flipping, interval algorithm.

1. Introduction

When we flip a coin, if it turns heads with a probability p , then we call it a p -coin. If $p = 1/2$, then we call the coin *unbiased*. In this paper, we address the problem of *coin flipping*: given two real numbers $p, r \in [0, 1)$, simulate an r -coin using a p -coin. The p -coin is called *source*, and the r -coin is called *target*.

The trick of von Neumann [10] is a classical example of coin flipping problem. Let H and T be the events for heads and tails, respectively. Flip the biased coin twice; if the result is HT (respectively TH), then output 0 (1), otherwise, (the events HH and TT) ignore the result and repeat the process. The probabilities that we get 0 and 1 are the same as $\Pr[HT] = \Pr[TH] = pq$. Hence, this method simulates an unbiased coin. Note, in this case, that von Neumann's trick works regardless of the source bias p .

Coin flipping is a subproblem of *random number generation*, in which we are concerned with generating an m -valued distribution $\mathbf{r} = \langle r_1, \dots, r_m \rangle$ using an n -valued distribution $\mathbf{p} = \langle p_1, \dots, p_n \rangle$ as source. Most of early works on random number generation focused on the case where the source distribution is unknown, including von Neumann's trick. Elias [2], Hoeffding and Simon [4], Stout and Warren [9], Peres [8], Dijkstra[1], and Pae and Loui [7] improved or generalized von Neumann's method in the case where the source distribution is unknown.

When the source distribution is *known*, of course, we can pretend that we do not know the distribution and apply one of the above methods. However, we should be able to take advantage of the knowledge of source distribution to improve in terms of the efficiency in usage of source.

Knuth and Yao [5] presented an elegant method to generate an arbitrary discrete probability distribution using an unbiased coin as a source and proved that their method is optimum in terms of the required average number of source coin flip to generate a target distribution.

As a first solution to the random number generation in the full generality, Han and Hoshi [3] proposed *interval algorithm*, which regards the source and target distributions as partitions of a unit interval and solves the problem, roughly speaking, by iteratively subdividing the target partition and approximating the target intervals by unions of small source subintervals. Moreover, their method generalizes Knuth-Yao method: in the case that the source is an unbiased coin, the interval algorithm reduces to the Knuth-Yao method. Hence, when the source is an unbiased coin, the interval algorithm is optimum. However, when the source distribution is not uniform, the interval algorithm does not seem to be optimal. In fact, Han and Hoshi [3] mention possible improvements of their method by rearranging the order of the subintervals and leave it as an open question. In their paper, the order of approximating subintervals of source distribution is fixed over all the iteration steps, and the improvement by changing the order seems very plausible.

In this paper, we investigate this question of improving interval algorithms in the context of coin flipping. We will present three versions of interval algorithm including the plain Han-Hoshi version. We present experiment results and compare the performances in terms of the average number of source coin flips, which we call the average cost, to generate one output.

2. Coin Flipping with Han-Hoshi Interval Algorithm

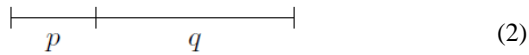
In this section, we discuss interval algorithms for the coin flipping problem for generating r -coin using p -coin. The symbols p and r are always the source bias and target bias, respectively. We also fix notations for tails probability for the source and target as follows: $q = 1 - p$ and $s = 1 - r$. Also, assume that $p < 1/2$.

Han-Hoshi interval algorithm starts with a unit interval partitioned in proportion to the target distribution. So in

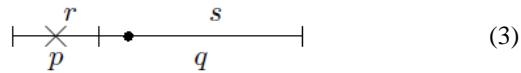
our problem of coin flipping, we consider an interval partition $[0, r) \cup [r, 1]$ as follows:



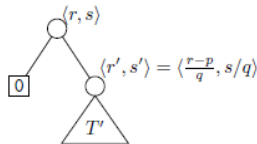
We will call the two subintervals of the partition r-interval and s-interval, respectively. Now consider another partition made of source subintervals, $[0, p) \cup [p, 1]$:



Again, we call these subintervals p-interval and q-intervals, respectively. In the above examples, the p-interval is included in the r-interval. In this case, the included interval is crossed out, and the remaining q-interval is partitioned further into two subintervals $[p, r)$ and $[r, 1]$ and they become the target interval for the next iteration:



In terms of coin flipping, this process corresponds to the following tree:



The tree represents a coin flipping process “flip the source coin; if it turns heads, output 0; otherwise, flip again and process according to the right subtree T' .” The left edge is taken for heads and the right edge is taken for tails. Now the subtree T' corresponds to the subdivision starting from the remaining q-interval in (3), which is partitioned into the ratio $r - p : s$. From this point, we solve the coin flipping problem for the new target $(r - p)/q$ with the same source p . Han-Hoshi interval algorithm proceeds by repeating subdivision as in (3) with a new target resulting from the previous iteration.

Depending on the values of p and r , Han-Hoshi subdivision considers three different cases shown in Figure 1. We call them *rules* for the plain interval algorithm, and the three rules completely describe the subdivision

algorithm together with the iteration process explained above.

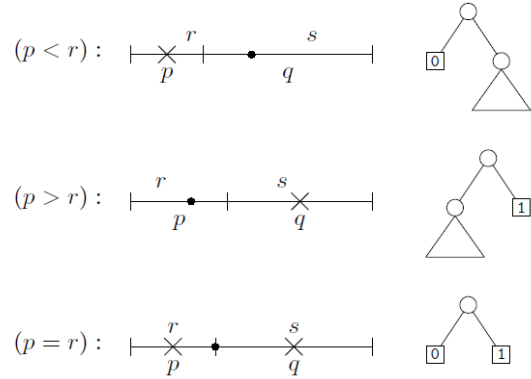


Fig. 1 The plain Han-Hoshi subdivision rules.

In case $p > r$, the q-interval is crossed out, and the remaining p-interval becomes the next target interval. If $p = r$, the target partition and source partition are the same and both source subintervals are crossed out, and the interval algorithm terminates. In this case, “flip the source coin; if it turns heads, output 0; otherwise, output 1.”

In order to see how this algorithm works with a real example, consider the case where $p = 1/3$ and $r = 1/2$. Figure 2 shows the subdivision process.

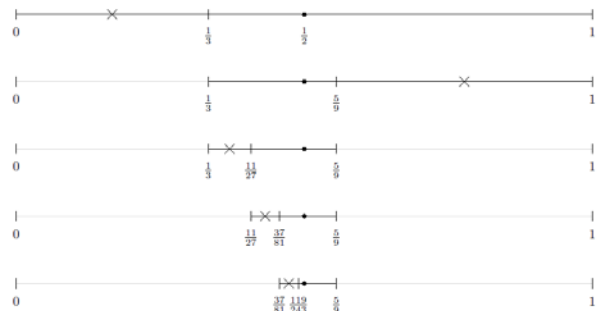
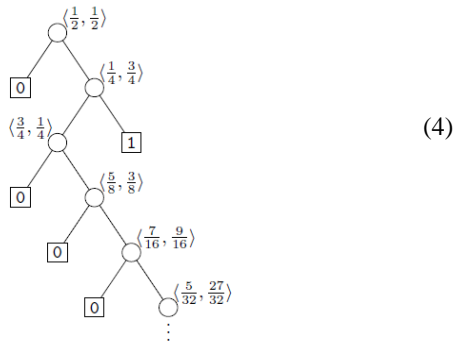


Fig. 2 Han-Hoshi subdivisions for $p=1/3$ and $r=1/2$.

In the example shown in Figure 2, the target subintervals are $[0, 1/2)$ and $[1/2, 1]$ and they are approximated by the union of source subintervals in each iteration, that is, $[0, 1/2) = [0, 1/3) \cup [1/3, 11/27) \cup [11/27, 37/81) \cup [37/81, 119/243) \cup \dots$, and $[1/2, 1] = [5/9, 1) \cup \dots$. The approximating subintervals for $[0, 1/2)$ correspond to the

output 0, and the size of the approximating subinterval is the probability of the output event occurs. Hence, the probability that the corresponding coin flipping process outputs 0 is 1/2. The same holds for the other starting target subinterval [1/2, 1]. The corresponding tree is as follows:

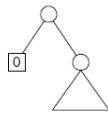


The *average cost* of coin flipping is the average number of source coin consumed to generate one output. If the coin flip process is represented as a tree, then the average cost is the average depth of the tree in which the left edge is weighted by p and the right edge q. The average cost of tree (4) is about 2.25. Note that, in view of the plain interval algorithm rules in Figure 2, the output 0 occurs only on the left edge (heads of the source coin), and output 1 always occur on the right edge (tails of the source coin).

3. Refinements of Interval Algorithms

3.1 Algorithm A

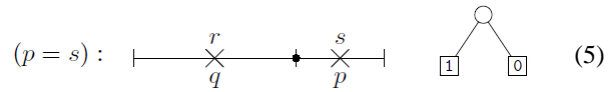
An improvement that we can make on the plain interval algorithms is as follows. Consider the case $p = s$ (and thus $q = r$). Since, in that case, $p < r$, the plain interval algorithm will result in a tree of the form



and its average cost is larger than 1 depending on the value of p. However, the following tree is clearly a better choice since its average cost is 1:

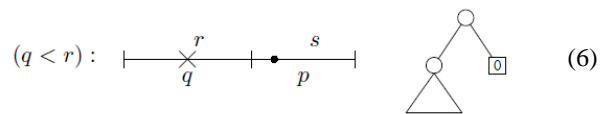


In terms of the interval subdivision, we can represent this rule as follows:



and it refines the plain interval algorithm. Note, in this rule, that the order of source subintervals is reversed. Similarly, the 0 output occurs on the right edge of the corresponding tree.

Now let us consider another rule that takes advantage of reversing the order of the source subintervals. When $q < r$ (therefore, $r > s$), we can cross out q-interval, instead of p-interval from r-interval:



This rule again refines the plain interval algorithm. Again, the corresponding tree has an output 1 on the left edge, as opposed to the plain interval algorithm.

Now, we define Algorithm A to be a new interval algorithm that consist of the rules of the plain interval algorithms in addition to the rules (5) and (6).

Figure 3 shows the subdivision process by Algorithm A for the same source and target as in Figure 2. We can view that rule (6) is better because it crosses out the bigger subinterval whenever it can, hence resulting in smaller average cost. Indeed, as we can see in Figure 3, the target intervals shrink more quickly.

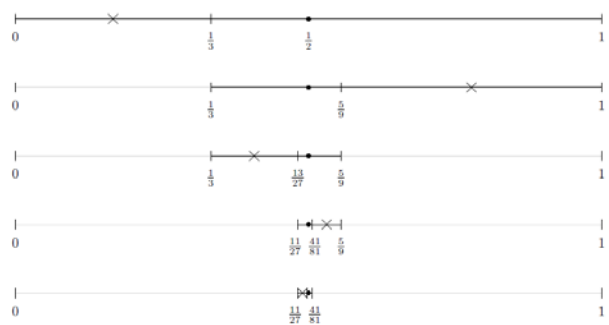
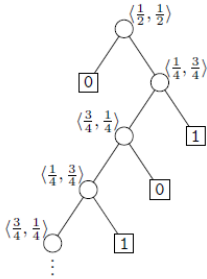


Fig. 3 Subdivisions by Algorithm A for $p=1/3$ and $r=1/2$.

The resulting tree from the subdivision process in Fig. 3 is as follows:



Its average cost is 15/7, which is smaller than that of (4).

Figure 4 shows the average costs of the resulting trees from the plain interval algorithm and Algorithm A for $p = 1/3$ and $0 < r < 1$. The dashed line is for the plain interval algorithm, and the performance of Algorithm A is always better or equal. Although the result is for $p = 1/3$, we can show similar result for other values of p .

Note that the average cost of Algorithm A shows a fractal-like behavior, while the graph for the plain interval algorithm is a straight line. The graphs were drawn by computing the average cost for the values of $r = k/2^{10}$, $k=0,1,\dots,2^{10}$. There are values of r , for example $1/3$, that results in average depth 1. But the graph does not show those values, and if the graph of the average cost of Algorithm A is indeed a fractal, then it will be impossible to draw all the infinitely many discontinuities correctly. We will see a similar behavior with Algorithm B that we discuss below.

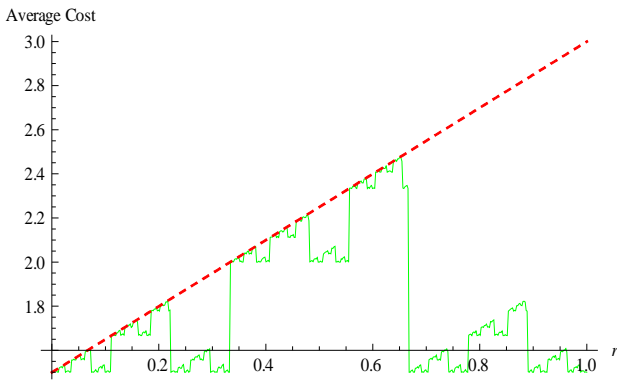


Fig. 4 Comparison of plain interval algorithm and Algorithm A.

3.2 Algorithm B

Now consider the rules in Figure 5 that replace the rule of the previous two algorithms, the plain interval algorithm

and Algorithm A, for the case ($p < r$). The resulting algorithm, which we call Algorithm B, crosses out the p -interval when p is smaller than both target subintervals. When $r \leq s$, the effect is the same as the rule that is replaced. But when $r > s$, the p interval removes part of the s -interval. By doing so, the next target distribution has the smaller entropy compared to the other case. Although this heuristic seems to work for many cases but it does not always result in subdivision process with smaller average cost than Algorithm A.

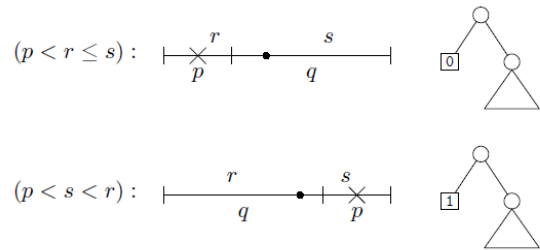


Fig. 5 New rules for Algorithm B

Figure 6 shows the performance comparison of the plain interval algorithm and Algorithm B for $p = 1/3$ and $0 < r < 1$. Algorithm B shows superior performance.

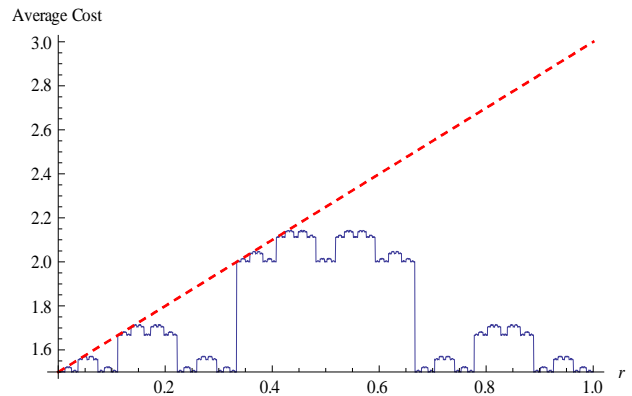


Fig. 6 Comparison of plain interval algorithm and Algorithm B.

An apparent observation is that the average cost of Algorithm B is symmetric with respect to the target bias r . In terms of the average cost with respect to the target bias,

the plain interval algorithm is totally asymmetric, and Algorithm A is more symmetric but not fully symmetric. An explanation comes from the symmetry of the rules employed by the algorithms. The rules of the plain interval algorithm are asymmetric with respect to r . And two additional rules of Algorithm A recover the symmetry of the rules of the plain interval algorithm. And the rules of Algorithm B fully recover the symmetry.

Figure 7 shows the performance comparison of the Algorithm A and Algorithm B. The latter performs generally better, although there are cases where Algorithm A is better.

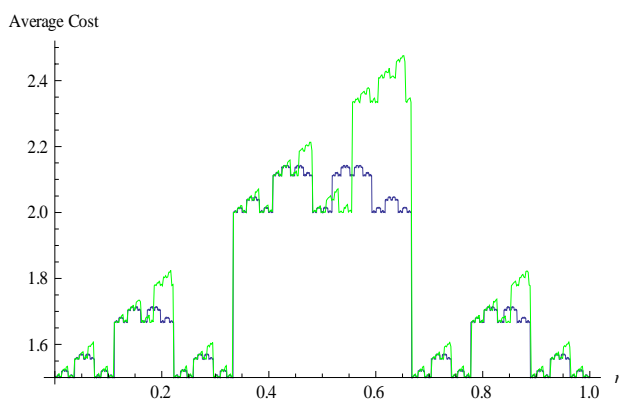


Fig. 7 Performance comparison of Algorithm A and Algorithm B.

4. Conclusion and Remarks

We discussed Han-Hoshi interval algorithm in the context of coin flipping problem. We presented two refinements of the original plain interval algorithm and demonstrated that both algorithms perform better in terms of the average cost. If we are to choose among the two algorithms, Algorithm B is probably a better choice since it generally performs better. Although we presented the experiment results for the case $p = 1/3$, we reach the same conclusion from more experiments that shows consistent results.

The idea that we employ to get Algorithm A and B, to improve upon the plain interval algorithm, is that it is better to cross out q -interval whenever possible. In doing so, we change the order of the source subintervals if necessary. To improve upon Algorithm A, Algorithm B removes p -interval from a smaller target subintervals, which results in a smaller entropy of the next target. While the first idea always improves the performance, the second idea does not guarantee improvement though it gives better results in general. These two ideas provide guidelines for what Han and Hoshi's original work [3] have left as a future work, namely, the problem of how to

suitably permute the subintervals to get a better performance.

Although we can improve the performance of the interval algorithms, the question of optimality still remains. As we observed, neither Algorithm A nor B is optimal. We can possibly get a better interval algorithm, for at least a fixed value of p , say $1/3$, by identifying the ranges of r where Algorithm A is better than Algorithm B, and then employing the better algorithm for given r . The resulting algorithm will be a further refinement of both, and as an interval algorithm it will have much more elaborate rules. Still, we do not know whether it will be optimal. In fact, Pae and Loui [6] showed that it is impossible to get an optimal coin flipping using branching decision based on algebraic computation. So the above refinement cannot be optimal, and by rearranging the order of subintervals we cannot obtain an optimal interval algorithm.

Acknowledgment

This work was supported in part by a Hongik University research grant and National Research Foundation of Korea Grant funded by the Korean Government (2009-0077288).

References

- [1] Edsger W. Dijkstra. Making a fair roulette from a possibly biased coin. *Information Processing Letters*, 36:193, 1990.
- [2] Peter Elias. The efficient construction of an unbiased random sequence. *The Annals of Mathematical Statistics*, 43(3):865–870, 1972.
- [3] Te Sun Han and Mamoru Hoshi. Interval algorithm for random number generation. *IEEE Transactions on Information Theory*, 43(2):599–611, 1997.
- [4] W. Hoeffding and G. Simons. Unbiased coin tossing with a biased coin. *The Annals of Mathematical Statistics*, 41:341–352, 1970.
- [5] Donald E. Knuth and Andrew C-C. Yao. The complexity of nonuniform random number generation. In Joseph F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results. Proceedings of a Symposium*, pages 357–428, New York, NY, 1976. Carnegie-Mellon University, Computer Science Department, Academic Press. Reprinted in Knuth's *Selected Papers on Analysis of Algorithms* (CSLI, 2000).
- [6] Sung-il Pae and Michael C. Loui. Optimal random number generation from a biased coin. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1079–1088, January 2005.
- [7] Sung-il Pae and Michael C. Loui. Randomizing functions: Simulation of discrete probability distribution using a source of unknown distribution. *IEEE Transactions on Information Theory*, 52(11):4965–4976, November 2006.
- [8] Yuval Peres. Iterating von Neumann's procedure for extracting random bits. *Annals of Statistics*, 20(1):590–597, 1992.

- [9] Quentin F. Stout and Bette Warren. Tree algorithms for unbiased coin tossing with a biased coin. *Annals of Probability*, 12(1):212–222, 1984.
- [10] John von Neumann. Various techniques for use in connection with random digits. Notes by G. E. Forsythe. In *Monte Carlo Method, Applied Mathematics Series*, volume 12, pages 36–38. U.S. National Bureau of Standards, Washington D.C., 1951. Reprinted in von Neumann's *Collected Works 5* (Pergammon Press, 1963), 768–770.



Sung-il Pae received the B.S. degree in Mathematics from Seoul National University in 1993, M.S. degree in Mathematics from University of Illinois at Urban-Champaign in 1997, and Ph.D. degree in Computer Science from University of Illinois at Urban-Champaign in 2005. During 2005-2007, he stayed at Korea Institute for Advanced Study before

he joined Computer Engineering department of Hongik University, Seoul, Korea. His research interest includes Algorithms and Theoretical Computer Science.