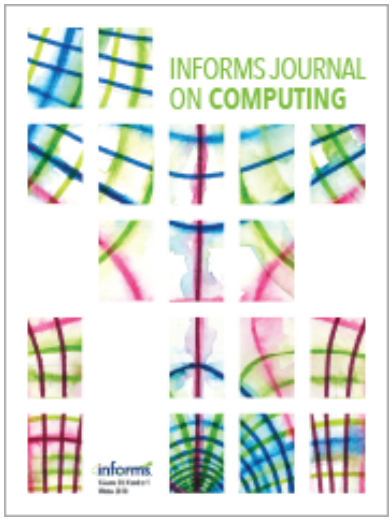

INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Commentary—Simulation of Algorithms for Performance Analysis

Pierre L'Ecuyer,

To cite this article:
Pierre L'Ecuyer, (1996) Commentary—Simulation of Algorithms for Performance Analysis. INFORMS Journal on Computing 8(1):16-20. https://doi.org/10.1287/ijoc.8.1.16

INFORMS Journal on Computing
Vol. 8, No. 1, Winter 1996

0899-1499/96/0801-0016 $01.25

COMMENTARY

Simulation of Algorithms for Performance Analysis

PIERRE L'ECUYER / *Département d'Informatique et de Recherche Opérationnelle (IRO), Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal H3C 3J7, Canada; Email: lecuyer@iro.umontreal.ca*

This feature article illustrates, using examples and anecdotes, how to use simulation experiments to better understand the behavior of computer algorithms and build *mechanistic models* of their performance. It also surfs over the vast amount of related scientific literature. The main question is how to use simulation in an efficient and reliable manner not only to estimate the (average) performance of different algorithms for certain classes of problems, but also to construct *metamodels* giving functional relationships between the performance measure of interest and parameters of either the algorithms or the class of problems considered (such as, for example, the problem size, the density of a graph, or the condition number of a matrix).

Simulation is a popular and powerful approach for studying the performance of complex stochastic systems of all sorts. It is used in diverse application areas such as manufacturing and inventory management, computer and communication systems, health-care management, transportation, military operations, marketing, finance, and so on. Analysis of algorithms is yet another application. Analytical tools are limited by the strong (and often unrealistic) assumptions they require. One then needs empirical methods, which should be based on sound scientific principles and (ideally) clever enough to give new insight and better understanding about the situation of interest.[17, 37]

A large body of theory and tools now exist (and more are being developed) for modeling, uncertainty representation and random number generation, efficiency improvement, statistical analysis, sensitivity analysis, stochastic optimization, graphical animation, and so on, in the context of stochastic discrete-event simulation. Much of this could apply to the simulation of algorithms as well as to other application areas, although more work is needed on specific algorithm simulation applications.

In the remainder of this commentary, I will mention some of the issues that are currently drawing the most attention from the research and applications perspectives. I do not pretend to give a literature survey, but I go beyond the coverage of McGeoch and give pointers to several recent contributions and surveys in which the readers can find pertinent references.

1. Modeling

For the simulation of algorithms, we might distinguish two levels of modeling: (1) characterizing and modeling the class of problems of interest and (2) modeling the algorithm itself.

Level 1 is crucial whenever one is interested in an *average* (or in the distribution of some) performance measure of an algorithm over a certain class of problems. For the average (i.e., mathematical expectation) to be meaningful, the class of problems must be characterized and a probability distribution over the problem instances in the class must be well-defined. Since the simulation results could be extremely sensitive to the choice of that distribution, it should be selected carefully in such a way that the model be truly representative of the class of problems of interest. In particular, using the uniform distribution is rarely justified, because the problems arising in real-life situations rarely behave like problems generated from the uniform distribution. Real-life problems often tend to have a special structure, and the probability distribution should be mostly concentrated over those problem instances with the right structure.[37] Finding an appropriate distribution is generally the hardest part of a simulation study. In her paper, McGeoch recognizes but quickly skips over that important issue.

Modeling the algorithm itself (level 2) means that instead of just running the (programmed) algorithm as is on the problem instances, one could also build a simplified model of the algorithm, which will capture all of its important performance characteristics, and whose implementation will be faster or easier to run. Instead of measuring the CPU time used by the actual algorithm on a given computer, one would measure other statistics such as, for example, the number of operations of a certain type, the quality of the solution obtained, etc.

The modeling issues discussed above are similar to those encountered in other application areas. For example, to simulate a high-speed communication network, one must model the structure of the network (which is easy) and the incoming traffic (which is harder, since information cells to be transmitted generally arrive according to complicated autocorrelated stochastic processes, often in bursts). That corresponds to level 1 above. At the second level, one models the algorithms that are used for routing, buffering, ordering (perhaps with priorities), and discarding the information cells. A large chunk of the knowledge developed for stochastic discrete-event simulation modeling, in general, is also pertinent to the setup of simulation of algorithms. See [3, 5, 18] for references.

It appears that several people evaluating the performance of algorithms tend to adopt rather arbitrary models for the

Subject Classifications: Computational experiments, algorithm design, algorithm development.

classes of problems they consider. In contrast, a lot more effort is usually made, in other areas of applications, to come up with more realistic models. For example, tremendous efforts are being made to build realistic models of the incoming traffic process in communication networks, of the machine failures and incoming demands in a manufacturing system, or of the customer behavior in fast-food restaurants.

Sometimes, the stochastic aspects of the system to be simulated are so complicated (for example, the dependencies so numerous) that building a realistic model is practically out of reach. An alternative in that case, when enough data are available, is *trace-driven simulation*, which basically consists in just replaying what has been observed in past history for some aspects (such as the demands in a production/inventory system or the incoming patients in a healthcare unit) and comparing the performance of different decision policies over that input. That approach has the advantage of capturing the complicated dependencies between different random variables without modeling them explicitly. A major drawback, on the other hand, is that the quantity of available data limits the total simulation length (or the number of runs) that could be performed. Hybrids between trace-driven and stochastic modeling approaches go around that problem. The analog of trace-driven simulation in the context of algorithms is to compare different algorithms on a given set of test problems collected (hopefully) from real-life applications. A hybrid approach here could partially *randomize* the test problems.

The author of this feature article explains that her approach is to build *mechanistic models*, "based on partial understanding of the process," which "provide a much safer basis of extrapolation beyond the range of experiments, and can give more parsimonious descriptions of results," in contrast to *explanatory rationales* based on *empirical models*, as used by Hooker,[17] "which make no assumptions about the process that produces the data." After looking at the worked-out examples, I must say that I do not see much difference between the methodology employed by McGeoch (for example, for the bin-packing example) and what is proposed by Hooker. Obviously, a model which provides better understanding of the process, and/or is simpler, is generally preferable. That stands for all areas of applications and is not new. Hooker advocates "using experiments to suggest hypotheses about the behavior of algorithms" and insists that "empirical science involves theory." He states: "The hypothesis is then tested empirically, using time-honored techniques of experimental design and statistical analysis. Eventually, one may put together a unifying and coherent picture that appears to explain why certain factors are important. This is a *theory*." The theory could be further tested empirically, and so on, much like in experimental physics and other natural sciences. In my view, McGeoch and Hooker just employ different words to express the same ideas. I agree with their ideas.

2. Statistical Analysis

This feature article merely scratches the surface of the main statistical issues and tools used in simulation. Elementary statistical methods, based on assumptions of normality, independence, stationarity, and the like, are not always robust to gross departures from those assumptions, and such gross violations are frequent in simulation contexts. See [1, 20, 34, 35] for illustrations and references to more refined statistical tools for simulation output analysis.

The following simple example caricatures one (perhaps not so rare) reason for things to go wrong with elementary techniques.

Example 1. Suppose one has an algorithm and a class of problems with a probability distribution P over that class, with the following properties. With probability 10^{-6} the algorithm will need 10^6 seconds to solve the (random) problem instance, and with probability $1 - 10^{-6}$ (i.e., in most cases), it will need only one second. (Replace seconds by main operations if you prefer.) The expected performance of the algorithm here is $\mu = 2 - 10^{-6} \approx 2$. Suppose μ is unknown and is estimated as follows: Run the algorithm on n random problem instances generated independently using distribution P, let $\bar{X}$ be the average time to solve those problems, and let S^2 be the sample variance. Clearly, $E[\bar{X}] = \mu$ and $\mathrm{Var}[\bar{X}] \approx (10^6 - 3)/n$ can be estimated by S^2. For reasonably small n (say $n \leq 10000$), one can make the following observations: (1) $\bar{X}$ is far from following the normal distribution; (2) with high probability, one will pick only easy problem instances and obtain $\bar{X} = 1$ and $S^2 = 0$. So, with high likelihood, a standard confidence interval for μ will be much too narrow (have zero width)! Here, not only does the estimator have high variability (relative to its mean), but the variability of its variance estimator, $\mathrm{Var}[S^2]$, is huge. This is a simple illustration of a *rare event* situation, which could very well occur in a more or less hidden or disguised form in the context of algorithm simulation, and is generally pretty hard to deal with. *Importance sampling*, briefly discussed in the next section, is one possible way of attacking such a problem.

An important idea that this example also illustrates is that the average performance of an algorithm does not always tell most of the story. Variability or other statistical measures of the performance could be as important. Performance measures should take into account different aspects such as the quality of the solution as well as the CPU time to obtain it (these are generally correlated random variables). One may define a *utility function* which takes all of these aspects into account.[37]

A well-known and difficult problem in simulation is the estimation of a steady-state average cost over an infinite time horizon. One approach in that context is to perform a single (very long) simulation run and start collecting statistics only after a certain warm-up period, in order to reduce the bias of the estimator. To estimate the variance (and eventually compute confidence intervals), one may use a *batch means* approach.[20] Finding good and reliable ways of selecting the warm-up length and the batch sizes are still open problems: [13, 20] Other techniques for estimating the variance are studied in [1, 4] and references therein. Infinite-horizon models are sometimes appropriate in algorithm

simulation, for example, to study the amortized average cost of a given data structure.

Much research is currently being done, in discrete-event simulation, on the related topics of *sensitivity analysis* (or derivative estimation) and on *stochastic optimization* via simulation. The aim is to find efficient ways of estimating how sensitive is a performance measure of interest to different model parameters, or eventually to optimize the model with respect to some (or all) of those parameters. Much of what has been found so far could certainly be applied in some situations to algorithm simulation. In that context, the parameters could be either parameters of the class of problems considered or of the algorithm itself. References on those two topics can be found in [9, 10, 16, 21, 29, 30, 38].

3. Efficiency Improvement

Let μ be the unknown quantity (performance measure) that we want to estimate and X an estimator (a random variable) computed via simulation. We define the *bias*, *variance*, and *mean square error* (MSE) of X as $\beta = \mathrm{E}[X] - \mu$, $\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2]$, and $\mathrm{MSE}[X] = \mathrm{E}[(X - \mu)^2] = \beta^2 + \mathrm{Var}[X]$, respectively, where E is the mathematical expectation. The quality of an estimator is often evaluated by its MSE. Let C_X be the expected cost (for example, CPU time) for computing X. Common practice is to estimate μ by the average $\bar{X}$ of n independent and identically distributed (iid) copies of X. In absence of bias $\mathrm{E}[\bar{X}] = \mu$ and $\mathrm{MSE}[\bar{X}] = \mathrm{MSE}[X]/n$, while $C_{\bar{X}} \approx nC_X$, so $\mathrm{MSE}[\bar{X}]C_{\bar{X}}$ is (roughly) independent of n. That motivates the classical efficiency criterion of an estimator:

$$\mathrm{Efficiency}[X] = \frac{1}{\mathrm{MSE}[X] \cdot C_X}. \tag{1}$$

In that context, *efficiency improvement* means replacing the current estimator X by another estimator Y with better (larger) efficiency. That could be achieved by reducing either the variance, or the bias, or the expected cost C_X. If X and Y are both unbiased and have similar computational costs, then improving the efficiency means reducing the variance and one talks about *variance reduction techniques* (VRTs). However, efficiency can sometimes be improved by increasing the variance and/or the bias, as illustrated, for example, in [8] and [14]. For example, if X has half the variance but costs 10 times more to compute than Y, then Y is more efficient.

Equation (1) is justified by the fact that when there is no bias, the MSE decreases as the inverse of the computational budget, which is proportional to n. The statistical precision, as measured, for example, by the width of a confidence interval, then decreases as $n^{-1/2}$. In other words, one must multiply the computational budget by 100 to gain one additional digit of precision. This is the most standard (or *canonical*) convergence rate in statistics, which follows from the central limit theorem.

If there is bias, however, the right-hand-side expression in (1) is generally not invariant with n and often converges to zero as $n \to \infty$. In that case, more refined tools are required to compare the efficiency of estimators.[14] The exact form of the estimator could also vary with the size of the total available computational budget to make the right compromise between the bias and variance.[14, 29, 30] Consider a budget-dependent estimator whose value is $X(t)$ after spending a computing budget t. Suppose that

$$\lim_{t \to \infty} t^{\gamma} \cdot \mathrm{MSE}[X(t)] = \kappa \tag{2}$$

for some positive constants γ and κ. Glynn and Whitt[14] call γ and κ the *asymptotic efficiency rate* and *asymptotic efficiency constant* of X, respectively. Given two estimators, the one with the largest γ is asymptotically preferred, and in case of equality, take the one with the smallest κ. The canonical case corresponds to $\gamma = 1$.

Recent up-to-date surveys or important contributions on efficiency improvement (and variance reduction), which I recommend looking at, include [2, 11, 12, 15, 23, 33, 34, 41]. Among the important recent advances in that area are the better understanding of correlation-induction techniques (including control variates, antithetic variates, and common random numbers), different variants of conditional Monte Carlo, and significant advances in the *importance sampling* methodology, based largely on the theory of large deviations.

Importance sampling amounts to changing the probability laws that drive the model, in such a way that rare but expensive events occur more often. To get a valid estimator after changing the probability law, the original estimator is multiplied by an appropriate correction factor. That could reduce the variance by huge factors (without introducing any bias) if the new probability laws are well chosen. However, finding such good choices is generally hard. Recent advances have provided effective guidelines for changing the probability laws for certain classes of models of reliability and high-speed communication networks, where rare events are involved much as in Example 1, and where naïve simulation is extremely inefficient.[15, 33, 39, 40] Efficiency gains by factors of over a million have been observed.

4. Random Number Generation

Stochastic simulations on computers must rely on random number generators (RNGs), which are actually simple deterministic programs trying to *fake* randomness. The RNGs available in today's software environments and used on a daily basis are of extremely uneven quality. Some classes of generators which seemed to do the job well enough a few years ago are now outdated, mainly because with the relentless increase in easily available computing power, larger and more complex simulations are being performed which use enormous amounts of random numbers. Vectors of successive values (or nonsuccessive values, with specific spacings) produced by the most popular families of RNGs have a lot of regularity (or geometrical structure) over the entire period of the RNG. In some cases, those structural properties show up in the output and significantly bias the results. This has occurred several times recently, with generators which were thought to be very good (see, for example, the references in

[24]). In most of these cases, structural defects in the generators were later found by a deeper theoretical analysis (see, for example, [6, 7, 25, 32]).

Here are some recommendations for RNG users who want to avoid bad experiences such as those described in the anectodes of this feature article.

1. Do not use too simple RNGs, such as the linear congruential generators with modulus fitting in 32 bits or with power-of-two modulus,[4, 19, 20] simple Tausworthe or GFSR generators whose characteristic polynomial of the recurrence is a trinomial,[20, 24] or add-with-carry and subtract-with-borrow generators.[31] Those generators, and others,[24] have a too simple structure and fail several tests.[22] The so-called minimal standard (generator) proposed in [36] is among those and should be avoided for serious applications. Use instead more complicated recurrences, such as *combined* generators.[24, 26, 27]
2. The sequence produced by any (deterministic) RNG is periodic. The period length should be not only larger than the number of values that are needed, but *several* orders of magnitude larger, for reasons explained in [24].
3. Proposed generators should be based on sound theory. Do not use a generator unless its period length is known and its structural properties are understood to a reasonable extent. That seems to contradict the idea of randomness, but a "better the unknown than the devil we know" philosophy could lead to disaster. No generator can be proved foolproof; we have to live with that fact. When in doubt, try two or more generators of *different types* on your problem and compare the results.
4. Good RNGs should be portable across programming languages and machines, and always produce the same sequence when started from the same initial state (seed). The seed should also be controllable (it should be possible to store a seed and reset the generator to it later on).
5. Proper use of variance reduction techniques (such as common random numbers) is facilitated by the availability of a random number package with several generators, tools for jumping ahead or backwards in the sequence, antithetic switches, and so on. See L'Ecuyer and Côté[28] for a first version of such a package. More powerful packages, in which generators can be created as needed in an objected-oriented fashion and in practically unlimited number, are currently under development.

5. Conclusions

To complement McGeoch's article, I have discussed certain topics of current interest related to modeling, statistical output analysis, efficiency improvement, and random number generation, for stochastic discrete-event simulation. I gave additional references for those who want to dig further into these subjects. Applied work on how helpful those ideas and techniques are in the context of algorithm simulation is needed and encouraged. Situations where standard techniques (such as certain random number generators or confidence interval estimators) could yield wrong or misleading results should also be identified and better understood, in order to find safer alternative techniques if possible or at least provide specific warnings.

Acknowledgments

This work has been supported by NSERC-Canada grant #ODGP0110050 and FCAR-Québec grant #93ER1654. I thank Jean-François Cordeau for his careful reading of the manuscript.

References

1. C. ALEXOPOULOS. 1994. A Review of Advanced Methods for Simulation Output Analysis, *Proceedings of the 1994 Winter Simulation Conference*, IEEE Press, 133–140.
2. A.N. AVRAMIDIS and J.R. WILSON, 1995. Integrated Variance Reduction Strategies for Simulation, *Operations Research*. To appear.
3. O. BALCI, 1994. Validation, Verification, and Testing Techniques Throughout the Life Cycle of a Simulation Study, *Annals of Operations Research 53*, 121–174.
4. P. BRATLEY, B.L. FOX and L.E. SCHRAGE, 1987. *A Guide to Simulation*, 2nd ed., Springer-Verlag, New York.
5. R.C.H. CHENG, 1994. Selecting Input Models, *Proceedings of the 1994 Winter Simulation Conference*, IEEE Press, 184–191.
6. R. COUTURE and P. L'ECUYER, 1994. On the Lattice Structure of Certain Linear Congruential Sequences Related to AWC/SWB Generators, *Mathematics of Computation 62*, 798–808.
7. R. COUTURE and P. L'ECUYER, 1996. Orbits and Lattices for Linear Random Number Generators with Composite Moduli, *Mathematics of Computation 66*. To appear.
8. G.S. FISHMAN and V.G. KULKARNI, 1992. Improving Monte Carlo Efficiency by Increasing Variance, *Management Science 38*, 1432–1444.
9. M.C. FU, 1994. Optimization by Simulation: A Review, *Annals of Operations Research 53*, 199–248.
10. P. GLASSERMAN, 1991. *Gradient Estimation Via Perturbation Analysis*, Kluwer Academic, Norwell, MA.
11. P. GLASSERMAN and D.D. YAO, 1992. Some Guidelines and Guarantees for Common Random Numbers, *Management Science 38*, 884–908.
12. P.W. GLYNN, 1994. Efficiency Improvement Techniques, *Annals of Operations Research 53*, 175–197.
13. P.W. GLYNN and P. HEIDELBERGER, 1992. Experiments with Initial Transient Deletion for Parallel, Replicated Steady-State Simulations, *Management Science 38*, 400–418.
14. P.W. GLYNN and W. WHITT, 1992. The asymptotic efficiency of simulation estimators, *Operations Research 40*, 505–520.
15. P. HEIDELBERGER, 1993. Fast Simulation of Rare Events in Queueing and Reliability Models, *Performance Evaluation of Computer and Communication Systems*, Lecture Notes in Computer Science, vol. 729, L. Donatiello and R. Nelson (eds.), Springer-Verlag, New York, 165–202.
16. Y.-C. HO and X.-R. CAO, 1991. *Discrete-Event Dynamic Systems and Perturbation Analysis*, Kluwer Academic, Norwell, MA.
17. J.N. HOOKER, 1994. Needed: An Empirical Science of Algorithms, *Operations Research 42*, 201–212.
18. M.E. JOHNSON and M. MOLLAGHASEMI, 1994. Simulation Input Data Modeling, *Annals of Operations Research 53*, 47–76.
19. D.E. KNUTH, 1981. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 2nd ed.*, Addison-Wesley, Reading, MA.
20. A.M. LAW and W.D. KELTON, 1991. *Simulation Modeling and Analysis, 2nd ed.*, McGraw-Hill, New York.
21. P. L'ECUYER, 1991. An Overview of Derivative Estimation, *Pro-*

ceedings of the 1991 Winter Simulation Conference, IEEE Press, 207–217.

22. P. L'ECUYER, 1992. Testing Random Number Generators, *Proceedings of the 1992 Winter Simulation Conference*, IEEE Press, 305–313.
23. P. L'ECUYER, 1994. Efficiency Improvement Via Variance Reduction, *Proceedings of the 1994 Winter Simulation Conference*, IEEE Press, 122–132.
24. P. L'ECUYER, 1994. Uniform Random Number Generation, *Annals of Operations Research 53*, 77–120.
25. P. L'ECUYER, 1995. Bad Lattice Structures for Vectors of Non-Successive Values Produced by Some Linear Recurrences, *ORSA Journal on Computing*. To appear.
26. P. L'ECUYER, 1995. Combined Multiple Recursive Generators, *Operations Research*. To appear.
27. P. L'ECUYER, 1996. Maximally Equidistributed Combined Tausworthe Generators, *Mathematics of Computation*. To appear.
28. P. L'ECUYER and S. CÔTÉ, 1991. Implementing a Random Number Package with Splitting Facilities, *ACM Transactions on Mathematical Software*, 17, 98–111.
29. P. L'ECUYER and G. PERRON, 1994. On the Convergence Rates of IPA and FDC Derivative Estimators, *Operations Research 42*, 643–656.
30. P. L'ECUYER and G. YIN, 1995. Convergence Rate of Stochastic Optimization Algorithms with Budget-Dependent Bias. Submitted.
31. G. MARSAGLIA and A. ZAMAN, 1991. A New Class of Random Number Generators, *The Annals of Applied Probability 1*, 462–480.
32. M. MATSUMOTO and Y. KURITA, 1994. Twisted GFSR Generators II, *ACM Transactions on Modeling and Computer Simulation 4*, 254–266.
33. M.K. NAKAYAMA, 1994. Fast Simulation Methods for Highly Reliable Systems, *Proceedings of the 1994 Winter Simulation Conference*, IEEE Press, 221–228.
34. B.L. NELSON, 1990. Control-Variate Remedies, *Operations Research 38*, 974–992.
35. B.L. NELSON, 1992. Statistical Analysis of Simulation Results, in *Handbook of Industrial Engineering*, G. Salvendy (ed.), Wiley, New York, Chapter 102, 2567–2593.
36. S.K. PARK and K.W. MILLER, 1988. Random Number Generators: Good Ones are Hard to Find, *Communications of the ACM 31*, 1192–1201.
37. C.R. REEVES, 1993. *Evaluation of Heuristic Performance*, Wiley, New York, 304–315.
38. R.Y. RUBINSTEIN and A. SHAPIRO, 1993. *Discrete Event Systems: Sensitivity Analysis and Stochastic Optimization by the Score Function Method*, Wiley, New York.
39. P. SHAHABUDDIN, 1994. Fast Simulation of Packet Loss Rates in Communication Networks with Priorities, *Proceedings of the 1994 Winter Simulation Conference*, IEEE Press, 274–281.
40. P. SHAHABUDDIN, 1994. Importance Sampling for the Simulation of Highly Reliable Markovian Systems, *Management Science 40*, 333–352.
41. J.D. TEW and J.R. WILSON, 1994. Estimating Simulation Metamodels Using Combined Correlation-Based Variance Reduction Techniques, *IIE Transactions 26*, 2–16.