ORSA Journal on Computing
Vol. 2, No. 1, Winter 1990

0899-1499/90/0201-0094 $01.25

Computational Testing: Why, How and How Much

HARVEY J. GREENBERG *Mathematics Department, University of Colorado at Denver, Denver, CO 80204, BITNET: hgreenberg@cudenver*

(Received: October 1989)

This considers the issues associated with performing and reporting computational testing in the context of original research in operations research, particularly in its interface with computer science. The scope includes non-numerical computations, such as fundamental algorithms and information structures in new modeling languages, and many criteria besides speed of computation, like robustness and depth. Beginning with a foundation of why we perform computational testing, some principles are described that draw heavily from the works of others. Then, methods of how to perform computational testing are suggested, ranging from quantitative statistical techniques to qualitative factor analysis. Finally, the issue of how much computational testing is appropriate is considered in the context of guidelines for referees and editors who deal with such issues.

This addresses the issues of computational testing in the context of its role in reporting research results. The approach is very much influenced by the recent report of the Committee On Algorithms (COAL), of the Mathematical Programming Society, by R.H.F. Jackson et al.[9]. Here we give greater consideration for non-numerical algorithms, such as testing the quality of modeling languages.

Even when testing is performed on numerical algorithms, like optimization, speed of computation is not regarded here as the sole objective. Other concerns include accuracy of results and robustness. Even these terms may have different meanings, depending upon whether we are testing algorithms, models, languages, environments, or systems. Even within each of these objects, there remains variation—for example, numerical algorithms have different measurements of accuracy and robustness than non-numerical ones. We emphasize that, unlike the ORSA guidelines,[16] we consider purposes of computational testing beyond the accompaniment of algorithms. Some of these additional reasons for computational testing are addressed by Zimmermann.[17]

The rest of this report is divided into three sections. In the next section we elaborate upon reasons for performing and reporting computational testing. At the same time, some principles are given and some are tacitly assumed. Section 2 suggests broad methods of testing, recognizing strengths and weaknesses of each. Finally, Section 3 gives guidelines for how much computational testing is appropriate, depending on why it is being performed. This comprises guidelines for referees and editors for the *ORSA Journal on Computing*.

1. Why Perform Computational Testing

The reasons to perform computational testing in the context of conducting and reporting research are to demonstrate:

- correctness of model or algorithm
- quality of solution
- speed of computation
- robustness.

These are, of course, not exhaustive, but they are enough to serve our purposes.

The extent to which one might report computational tests that suggest mere correctness is difficult to quantify. Certainly such testing takes place in any algorithm implementation. When it is straightforward, the computational tests are not reported (or they are enhanced to satisfy another goal, like demonstrating speed of computation). There are, however, some cases where correctness is an issue that cannot be settled, and computational testing does offer some illumination.

One example is the neural net proposed by Hopfield and Tank[8] to solve the Travelling Salesman Problem (TSP). The dynamics of the penalty trajectories require more analysis, and their paper was about the feasibility of building a neural computer (hardware) to solve the TSP, not about how well it does it. Properties of theoretical convergence (say by statistical considerations like simulated annealing) are not well understood at present, so empirical results are important to add insights into the neural dynamics. Similar situations prevail for other algorithm trajectories.

The case of model correctness is also part of this. One good example of using computational testing to address model correctness is Manne's introduction of the refinery linear programming model.[12] After presenting a detailed description of assumptions and mathematical relations, computed shadow prices are compared against actual market prices. Favorable comparison suggests that the model represents refinery operations rather well. Formally, such validation exercises

do not prove a model's accuracy, but well-designed tests do suggest credibility.

One of the essential purposes of computational testing in the simulation community is to address the issue of model correctness (see Balci and Sargent[1] for a survey). Early concerns with language seem not to be addressed, at least in the present context. Random number generation is another topic that had computational testing. New aids for incorporating artificial intelligence into the modeling environment raises new concerns of quality of model generation, which can be addressed by computational testing.

Furthermore, there are new concerns for model correctness raised by parallel computation. Nance[14] had studied time flow mechanisms and reported computational tests for the machine interference problem as a way to illustrate the issue of how to represent concurrence (for example, not registering a departure from an empty queue). This is now a central issue at the crux of parallel simulation, where there is a risk of *overparallelization* to gain speedup. The result may not be valid if the implementation does not limit parallelization to what is natural for the model, especially for analog computation.

Another problem in simulation is the sample size. A recent note by Harris[6] reflects both the importance and the scientific care that must be taken. Other parameters may be best studied empirically; and, even when this leads to theoretical (or analytical) results of how to set certain control parameters, the computational tests that led to that discovery may be important to report. Another example of sharing empirically derived insights is in the well orchestrated study by Glover et al.[4]

2. How to Perform Computational Testing

Methods of computational testing include:

- *statistical analysis*—presumes random generation over a problem space and collects performance values of replicated trials.
- *library analysis*—uses a fixed library generally available to the professional community and whose properties are already known or are reported along with the computational test results.

An early example of statistical analysis to evaluate performance is the use of Analysis Of Variance by Moore and Whinston[13] to measure significance of algorithm parameters and problem attributes against computational time and iterations. A more recent analysis was given by Hoaglin et al.[7] Generators, such as NETGEN,[10] enabled some degree of randomization while retaining realistic structural properties.

In attempting to use CPU time as a measure of goodness, Gilsinn et al.[3] reported a pitfall when running in a multitasking environment. They showed significant variation of times due to job mix and time of day (see also O'Neill[16]).

The critical thing about using statistical analysis for any purpose is the design of the experiment. In most (perhaps all) cases random generation of the coefficients without regard for problem structure is rather meaningless, certainly for any alleged demonstration of quality or speed. Instead, one may generate a particular model, like production scheduling and distribution, and randomize some of the basic data, such as costs, within realistic ranges. The *Workbench for Research In (linear) Programming*[5] (WRIP) is designed to contain this capability in its model generation/variation language.

Library analysis has always been popular in demonstrating merits of mathematical programming algorithms. Its main virtue is familiarity among the professional community. One recent example is the NETLIB linear programs put into public domain by Gay,[2] and thoroughly analyzed by Lustig.[11]

It is important to note that a library should reflect a reality, such as collected from industries. While "toy problems" are sometimes useful for exposition (and may be encouraged for just that reason), they seldom offer credibility in the present context of computational testing. If some industrial problems are used as paradigms but scaled due, for example, to budget constraints, problem size should still be realistic if inferences about actual performance are to be credible. The meaning of "size" may depend upon the situation; and, for non-numeric applications this may be difficult to separate from complexity. An example of the latter is to evaluate a *x-by-example* approach, where x may be *query* or *model*.

One disadvantage of a library, even one as extensive as NETLIB, is the inability to make inferences in some rigorous manner. This may be overcome (as in WRIP) by taking a particular problem, such as a library member, and allow controlled random variations. One form of variation is perturbation of the model's numerical values, keeping fixed the topology (as the sign pattern of an LP matrix). Another variation is augmentation of special objects or relations (such as redundant or infeasible rows in an LP).

In short, controlled randomization is a valuable method for satisfying the need for computational testing. The "control" must be part of the experimental design, which could target for certain classes of problems.

The issues of overparallelization, cited above in the context of concurrence in simulation, embody broad issues of scales and measurements. Presently, Richard Barr is conducting a study of how to report speedup in this regard. It is not as straightforward as one might at first imagine. Indeed, if the entire subject

of computational testing is in need of research, one focus that has received little attention due to its newness is in parallel computation in all areas of operations research.

3. How Much Computational Testing to Perform

In considering how much computational testing to perform, one must classify the purpose by its need:

Critical—the merit of the research contribution depends critically—perhaps entirely—on the empirical evidence provided by the computational testing.

Decisive—the merit of the research contribution depends decisively on the computational test results, but there is merit without it.

Valuable—the merit of the research is enhanced by the computational test results, but there is enough merit without it.

Incidental—the merit of the research is unaffected by the computational test results.

There is also the perennial point of proprietary software. If computational testing is critical or even decisive for the paper to endure as a contribution to the art and science of OR/CS, it must follow the same criteria. The author(s) have a greater burden, but it has been done while protecting trade secrets. Alternatively, if disclosure is not desirable, another way to inform the professional community is by a submission to the *Software Section* (both the *ORSA Journal on Computing* and *Operations Research Letters* have this).

If one looks at the full spectrum of outlets, there is the research paper at one end and paid advertisement at the other. Between these are articles in trade magazines (like *BYTE*), features in newsletters (like *CSTS Newsletter*), extended abstracts or demonstrations at professional meetings, and the list goes on. The present consideration is for a research article. For this there can be no compromise with necessity. Either computational test results are necessary for publication or they are not. If they are, there is no special dispensation for private owners to withhold vital information. The question then arises, "What about revealing the test problems and the exact code that was run?" This is really the issue of proprietary software in computational testing.

Our policy is:

> Any research paper whose merit depends critically (perhaps decisively) on the computational results must be prepared to have the results reviewed by referees.

In many cases this will not be invoked, but it must be understood that it may be (at the discretion of the Editor). Special steps may be taken to protect the owner's interests, but verification of scientific experiments is fundamental to the quality and integrity of our profession.

In closing, some elaboration of the meaning of "review" is in order. There are many variations possible, which will be negotiated between the author(s) and the Editor, but the intent is to ensure that the results reported satisfy independent verification. In some cases this may involve the authors' code being made available through the Editor (with prior assurance of appropriate protection). Under no circumstances will a paper be published in the *ORSA Journal on Computing* whose merit and information content depend critically on empirical claims that are not subjected to such review. The greater the claim, the more likely complete, independent testing will be required. The judgements of when, how and how much rest with the Editor.

ACKNOWLEDGEMENTS

The author gratefully acknowledges help from Saul I. Gass, Carl M. Harris, Richard H.F. Jackson, Jan-Karel Lenstra, Robert R. Meyer, Richard E. Nance, Ronald L. Radin, and Layne T. Watson. Part of this work was supported by ONR Grant No. N00014-88-K-0104.

REFERENCES AND BIBLIOGRAPHY

1. O. Balci and R.G. Sargent, 1984. A Bibliography on the Credibility Assessment and Validation of Simulation and Mathematical Models, *Simuletter 15:3*, 15–27.
2. D.M. Gay, 1985. Electronic Mail Distribution of Linear Programming Test Problems, *Mathematical Programming Society Committee On Algorithms (COAL) Newsletter*, December.
3. J. Gilsinn, K. Hoffman, R.H.F., Jackson, E. Leyendecker, P. Saunders and D. Shier, 1977. Methodology and Analysis for Comparing Discrete Linear L_1 Approximation Codes, *Communications in Statistics, Simulation and Computations B6:4*, 399–413.
4. F. Glover, D. Karney and D. Klingman, 1974. A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems, *Networks 20:5*, 191–212.
5. H.J. Greenberg and R.E. Marsten, 1988. An Experimentor's Workbench for Karmarkarian Analysis, Presented at the Fourth SIAM Conference on Discrete Mathematics (June 13–16), San Francisco, Calif.
6. C.M. Harris, 1987. Estimating Quantiles by simulation: A Brownian motion example, C291, *Journal of Statistical Computation and Simulation 28:3*, 266–272.
7. D.C. Hoaglin, V.C. Klema and S.C. Peters, 1982. Exploratory Data Analysis in a Study of the Performance of Nonlinear Optimization Routines, *ACM Transactions on Mathematical Software 8:2*, 145–162.
8. J.J. Hopfield and D.W. Tank, 1985. Neural Computation of Decisions in Optimization Problems, *Biological Cybernetics 52*, 141–152.
9. R.H.F. Jackson, P.T. Boggs, S.G. Nash and S. Powell, 1989. Report of the Ad Hoc Committee to Revise the

Guidelines for Reporting Computational Experiments in Mathematical Programming, *ORSA/CSTS Newsletter 10:1*, 7–14.
10. D. KLINGMAN, A. NAPIER and J. STUTZ, 1974. NETGEN: A Program for Generating Large-Scale Assignment, Transportation, and Minimum Cost Flow Network Problems, *Management Science 20:5*, 814–821.
11. I.J. LUSTIG, 1989. An Analysis of an Available Set of Linear Programming Test Problems, *Computers and Operations Research 16:2*, 173–184.
12. A.S. MANNE, 1958. A Linear Programming Model of the U.S. Petroleum Refining Industry, *Econometrica 26:1*, 67–106.
13. J.H. MOORE and A.B. WHINSTON, 1966. Experimental Methods in Quadratic Programming, *Management Science 13:1* (Series A), 58–76.
14. R.E. NANCE, 1971. On Time Flow Mechanism for Discrete Simulation, *Management Science 18:1*, 59–73.
15. R.P. O'NEILL, 1978. Comments preceding a Panel Discussion, Issues in the Evaluation of Mathematical Programming Algorithms, *Proceedings of the ACM Sigmap Bicentennial Conference on Mathematical Programming*, W.W. White (ed.), NBS Special Publication 502 (February), Washington, D.C.
16. Reporting Computational Experience in Operations Research, *Operations Research 27:1* (1979) vii–x.
17. H.-J. ZIMMERMANN, 1980. Testability and Meaning of Mathematical Models in Social Sciences, *Mathematical Modelling 1*, 123–139.

SUPPLEMENTARY BIBLIOGRAPHY

H.J. GREENBERG, 1989. Validation of Decision Support Systems, NATO ASI Series F48: *Mathematical Models for Decision Support*, G. Mitra (ed.), Springer-Verlag, Berlin, FRG.