scientific presentations also. If a speaker is unwilling to provide sufficient detail to back up computational claims, then the talk is advertising. Such talks should be grouped with other exhibits and sales presentations.

This is not to say that commercial concerns are henceforth to be barred from speaking and publishing. In fact there are many examples of successful efforts to continue in the scientific tradition of full disclosure, while still maintaining a competitive edge. In most of these cases, the success was based on publishing full accounts of the theoretical basis of a piece of software along with the details and results of the sound, well-thought-out, statistically based computational experiment, with documentation of the test problems used. In these cases, which can serve as paradigms, the only item withheld from the community was the code itself. Since the code usually represented many thousands of manhours of work, publishing the theoretical underpinnings did not compromise a competitive edge, and was good science. Furthermore, in all cases the developers were willing to submit their codes to be used in verification or comparison testing, even to other researchers hoping to prove that they could outperform them. In deciding a particular case, a referee must assess whether the level of detail provided is sufficient for a qualitative replication of the results. If not, then the author must either weaken the claims or provide further detail.

2.2 Replication, Verification, and Proof

In the opening paragraphs of this section, we mentioned the important role that the referee has in evaluating the sufficiency of computational results. In this part we wish to elaborate on this role.

With respect to computationally oriented papers, the referee's responsibility is to ensure the plausibility of the results, and that enough information is provided so that the experiment could be replicated with qualitatively similar results leading to the same conclusions. To be more clear, it is not generally the responsibility of the referee to repeat the experiment, run the code, or otherwise prove anything regarding claims made in a paper. (This general rule certainly does not apply when the software itself is to be published.) Verification, if ever performed, is provided by other scientists once the paper appears.

Since the issue for the referee rests so much with determining when sufficient information has been provided so that others may replicate results, we offer an example to illustrate sufficiency. Consider a paper that proposed a new method for some optimization problem. Suppose that paper described the algorithm and contained a few computational results to demonstrate feasibility of the idea and some exceptionally promising results. Such a paper would likely be publishable. If that same paper, however, went on to claim an order of magnitude improvement in speed over other methods, much more in the way of computational testing would be required. Also, much more in the way of algorithm description would be required. It would be necessary to decribe in detail the parts of the algorithm that make the implementation so much faster, e.g., the linear algebra. Without the latter, sufficient information is not available for replication, leaving others to rediscover the process before verification and progress can be achieved. This may be good marketing, but definitely not good science.

Another example is a paper that describes an algorithm for which, say, $\log(n)$ operations

were observed in practice, but for which the proof of $\log(n)$ has escaped the author. If the author has any theoretical results, say a proof of n operations, these of course should be included, along with the computational results. It remains for the referee to determine whether sufficient evidence is provided in the computational results to support a claim of $\log(n)$.

3 Measuring Computational Performance

In this section we consider the issues in designing and conducting the computational experiments that form the basis for the claims that are to be made. In particular, we discuss test problems and performance measures.

3.1 Choosing Test Problems

To make it easier to compare results from different sources, it is generally recommended that standard test problems be used. In this way, results in any particular paper can be more easily combined with results elsewhere to provide an extensive measure of the properties of a particular algorithm.

Defining a "standard test problem" is difficult. Typically a test problem becomes "standard" simply through widespread use. There are well known collections in several problem areas. For example:

1. nonlinear programming—Hock and Schittkowski [HocS81]
2. unconstrained minimization—Moré, Garbow and Hillstrom [MorGH81]
3. linear programming—the netlib collection [Gay85]
4. global optimization—Pardalos [Par87]

The deficiencies of specific collections of problems are well known. For example: The test set is relatively small compared with the total range of potential test problems. The problems are often small and regular so that they are easy to program. Various problems in a test set may have similar properties (for example, the Moré, Garbow, and Hillstrom set consists entirely of minimization problems that can be expressed as nonlinear least-squares problems). Optimizing an algorithm for a set of test problems may not give ideal performance in other settings.

There have been attempts to devise test problems and procedures with better diagnostic properties. For example, Lyness [Lyn79] suggested the interesting but computationally intensive technique of performance profiling. The use of such techniques may be illuminating, but it may also be unreasonable to impose such a constraint on a researcher.

If an algorithm has been selected to have a particular special property, then the test problems should illustrate this property clearly, and it may be necessary to devise or select new test problems for this purpose. Including new test problems that do not illuminate properties of the algorithm is not generally recommended.

3.2 Choosing Performance Measures

Performance measures have been proposed, discussed, debated, and sometimes vilified in the literature ([CroS80], [Bus80], [MieG78], [HofJ82]) since the first known computational evaluation of mathematical programming software [HofMSW53]. Despite the number proposed, they usually fall into four categories:

1. efficiency, which measures both the computational effort required to obtain a solution (e.g., CPU time, number of function evaluations, number of iterations), and the quality of the solution obtained (e.g., accuracy);

2. robustness, which measures how well a code can recover from improper input;

3. reliability, which measures how large a class of problems the code can solve; and

4. ease-of-use, which measures the amount of effort required to use the software (e.g., set-up time, quality of documentation, well-structuredness of the code).

(Robustness and reliability are sometimes used interchangeably, sometimes reversed in definition, and often confused.)

Regardless of what performance measures one uses, and what category they fall into, the reason for using them is to construct a number associated with each item tested (i.e., the computer implementations of the algorithms, or the codes), so as to summarize the performance criteria being investigated. It is important therefore that authors state clearly what is being tested, what performance criteria are being considered, and what performance measure is being used to draw inferences about these criteria. It is then the responsibility of the referee to judge whether the correct choices were made and whether enough information is being provided to make that judgement.

As part of this judgement, referees should bear in mind that performance measures are summary statistics, and as much as possible, should conform to all the accepted rules regarding the use thereof. Rather than attempt to summarize such rules, we encourage readers to consider [Hie81], [Hie82], [Gay83], and [HoaKP82] where careful testing is carried out.

3.3 Analyzing Results

When analyzing results from a computational experiment, one should bear in mind that one is indeed conducting an experiment. Thus one can do more to analyze computational results than simply reporting solution times, especially if one has many test problems, e.g., if one is using a test problem generator. Experimenters should consider the statistical nature of their efforts, and investigate the rich body of information available from that discipline. Even in situations where one has only a few results, there may be statistical tests that can provide useful insights, if only to put results in perspective. If there are none, performance claims could be suspect.

We have frequently found that some time spent with a good statistician before beginning an experiment has paid large dividends in the end. It is not at all uncommon to come away with new information on how to analyze results or perhaps redesign (or design if you

(i.e., that it "works"), then it is up to the referee whether the idea can stand on its own merits as a contribution to the literature. No elaborate computational experimentation is normally required if the idea can be evaluated on its own merits by the referee. If, on the other hand, a claim is made as indicated above, that this implementation is somehow better than another, evidence must be supplied to support the claim, and this is usually in the form of computational results. The referee must then judge whether enough computational evidence has been supplied to support the claim.

There will, of course, be cases where a referee might require more computational results even when no comparisons are made. In an increasingly computational world, this is sometimes, but by no means always, the only way to judge the value of a new idea, even an idea directly related to computation. For example, a complexity result stands on its own merit without any testing. Exceptions on the other side of the breakpoint cannot be sanctioned, however. That is, once a comparison is made, thorough computational evidence must be supplied.

This is, of course, not to say that any new idea is publishable so long as enough computational evidence is supplied to show that it "works." An example of this would be a new algorithm for small-scale unconstrained minimization with a few test runs to show that it finds a minimum. Such a paper is not likely to be published. In fact, most referees would send it back with the comment that it could only be published with convincing computational evidence of its value. On the other hand, a new idea for solving large-scale unconstrained problems that is shown to be computationally promising, would likely be publishable.

2.1 Proprietary Software

Proprietary software presents special problems, and therefore requires more discussion. Crowder et al. spent considerable time thinking about these problems and presented their views in their section on reproducibility. While we basically agree with their views and the guidelines provided there, we think that recent events argue for an update to, or refinement of, these views.

The original Guidelines attempted to reconcile the scientific method's requirement for full disclosure with a software vendor's desire to maintain proprietary rights by concentrating on reproducibility of results. In their paper, Crowder, et al. stated that "an absolute, reasonable, and scientifically justifiable criterion should be that the authors themselves be able to replicate their experiment." This is sound, but does not go far enough. A policy such as this addresses the need for replication, but ignores one of the main reasons for publishing research results: to inform others so that they may replicate, verify, modify, and build on one's results. This is the process by which scientific progress may be achieved. In the case of proprietary software, this last objective is violated intentionally. In the software marketplace, the guiding principal is to withold as much information as possible so as to gain a competitive edge. This usually translates into withholding, at the very least, the details of the implementation.

We are not opposed to making money. We do believe, however, that intentionally withholding information so that others may not follow in one's footsteps has no place in the world of science; such efforts belong in the world of marketing. This principle extends to

2 Performance Claims

An issue of primary importance in computational mathematical programming is performance claims. Although this issue is discussed in the original Guidelines [CroDM79], there has been some confusion over how much computational testing is required to support a given claim. In fact, there have been cases of researchers who declined to publish a new theoretical result in a journal that requested authors follow the Guidelines, because they believed that too much computational testing would be required. The papers went elsewhere. We hope to eliminate such confusion with this update to the original guidelines.

It is essential at this stage to have firmly in mind the types of claims that can be made and the computational evidence necessary to substantiate these claims. The goals of the work should dictate the type of implementation, the level of experimentation, and the depth of analysis of the raw data from the experimental runs. We have identified three types of computational studies. These are identified below along with their appropriate performance claims.

1. Preliminary testing to show feasibility and promise of a new algorithm or scheme. In this case, the code will typically be a "research" code not meant for general distribution. The results of its performance on several well-chosen problems would probably suffice.

2. More detailed experimentation to assess the strengths and weaknesses of an implementation. Here a range of problems must be appropriately chosen and more details of the implementation and algebraic overhead must be provided. Presumably, more care will go into the code, but it still may not be suitable for release.

3. Detailed comparison of the performance of an implementation with prominent methods already available. This is by far the most difficult type of experiment. Great care must be taken so that comparisons are fair. Performance measures (see §3.2) must be comparable for each algorithm in the test set, and the design philosophies of each code must also be forthrightly stated. For example, one might compare a program designed for stability, portability, and flexibility of use over a wide range of machines versus a limited implementation tuned to the idiosyncracies of a particular machine. In this case, a comparison of speed alone would be unfair, but a test demonstrating the cost incurred in using a more robust code would be perfectly appropriate.

The Guidelines suggested that it was the referee's responsibility to decide whether the empirical evidence given in the paper supported the computational claims made. That it will be left to the referee will likely always be the case, as discussed in §1. There are some simple rules that have been discussed often over the years and that we will repeat here to assist both authors and referees.

The principal issue seems to be in determining when a thorough, statistically-based computational experiment must be performed. In the past, it was argued that the proper breakpoint is where a paper contains a claim that a proffered implementation is better (faster, more accurate, more efficient, easier-to-use, etc.) than another. That is, if a researcher has a new idea, codes that new idea, and runs a few tests to demonstrate feasibility of the idea

were smart enough to have this conversation in advance) the experiment to obtain desired information with minimum effort.

In any event, referees again bear the responsibility for judging whether the techniques used to analyze results are appropriate for the data collected and the experiment designed.

4 Reporting of Computational Tests on Advanced Architectures

When reporting numerical tests performed on unusual computer architectures ("parallel" or "vector" computers), all of the above topics are of course relevant. In addition, further details should be provided to describe the nature of the calculations accurately. This information makes possible the accurate assessment of the results, and (given access to appropriate computers or simulators) might allow the results to be duplicated or emulated.

When computers with advanced architectures are used, the description of the hardware should mention those features that have enhanced or limited the performance of the algorithms or the range of problems solved. For multi-processor ("parallel") computers, these might include:

1. The number of processors—both the number available on the machine, and the number used in the calculation. If the architecture of the computer makes some processors more powerful than others, the author should mention which processors were used. (This is relevant for example, if the interconnection scheme is not uniform, or if certain processors have greater storage or computational power).

2. The interconnection scheme—ring, grid, hypercube, etc.

3. The memory layout—whether there is a global memory, the capacity of local memory on each processor (if this limits the algorithm or its proposed application).

4. Communication speed—the relative speeds of computation and communication; if timing is a factor in the test results, the absolute speeds of computation and communication may be relevant; it may also be important to mention transmission times for both one number and n numbers.

For pipelined ("vector") computers, the description of the hardware might mention:

1. The number of pipelines—both available and used.

2. Limitations on vector lengths—for example, on the Cray-1, vector pipelines take data from vector registers of length 64.

3. The number of paths to memory.

4. Pipeline speeds—timing reports should mention not only the maximum speed of a pipeline, but also the startup costs of using the pipeline, as well as the vector length required to achieve half the maximum speed (the "half-performance length," $n_{1/2}$ [HocJ81].

For computers that do not fall neatly into either of these categories, the guiding principles at the beginning of the section should be followed with the above lists serving as examples.

Currently, the use of advanced architectures implies a greater interaction between hardware and software than on sequential machines. When describing the implementation of an algorithm, the author should indicate how the special features of the machine were exploited.

The author should indicate how the special features of the hardware were used, whether through special programming techniques, through calls to hardware-optimized BLAS (Basic Linear Algebra Subroutines (Dongarra et al. [DonBMS79]), or via a special compiler. If the algorithm assumes a relationship between the number of processors and the size of the problem being solved, then techniques for mis-sized problems should be described.

The costs of an algorithm on a parallel computer are also more complex to describe. The operation counts should not just mention arithmetic costs, but should include the costs of communication among the processors. It may also be important to mention how data are passed from one processor to another, and whether global broadcasts are used.

When comparing a parallel algorithm with a scalar method, it is preferable to compare the parallel method not only with its scalar specialization, but also with the best scalar methods. In addition to reporting absolute speed-ups, times normalized by the number of processors are desirable.

Not all of these items will be relevant in all circumstances.

1. the results presented must be sufficient to justify the claims made;
2. there must be sufficient detail to allow reproducibility of the results; and
3. it is not the referee's duty to reproduce the results.

We mention at the outset that the profession cannot expect the refereeing process to do much more than to assure the plausibility of results. It is not the job of a referee to reproduce the results in computational sciences any more than a referee in any other experimental science to re-conduct the experiments. The referee must only be convinced that the first two principles are satisfied. In particular, a referee cannot be expected to expose fraud. This means that the system of conducting and reporting the results of scientific computations relies on the honesty and integrity of all concerned. Finally, we do not suppose that any such guidelines will interfere with the referee's main function of judging the scientific value of a proffered contribution. Papers which satisfy the above principles may simply contain a routine idea or results without any true significance, and thus not be worthy of publication.

As noted above, we do not think that a referee must reproduce the results to be convinced that the results are plausible. Some of the existing guidelines and some editorial policies require that a copy of the code and documentation used to obtain the results be made available to the referee on request. This is difficult to apply to proprietary codes and to research-level codes not suitable for use outside of the environment in which they were developed. More will be said on this topic in subsequent sections.

Honest mistakes can be made. Our literature, as well as the literature of other experimental sciences, is replete with examples of seemingly spectacular results that no one else can duplicate, or of "good" ideas that do not pan out. It is the responsibility of the profession in general to explore ideas and to distill the lasting contributions.

Many authors still confuse an *algorithm* with an *implementaton* of an algorithm. Roughly speaking, an algorithm is a problem solving template that leaves some practical details unspecified. It thus corresponds to a class of computer programs (its implementations), with certain sequences of instructions in each implementation corresponding to the steps of the algorithm. There are, for example, many implementations of the simplex algorithm, and they exhibit a vast range of computational performance.

Authors sometimes suggest that results for one implementation apply to *all* of an algorithm's implementations; one should regard such claims with considerable skepticism.

1.2 Outline of the Paper

In §2 we discuss various levels of claims that can be made concerning computational results. These range from the modest claim that a procedure "works" to a claim of overall superior performance. In §3 we consider testing and performance measures that are the raw data from which claims are made. Finally, in §4, the additional problems of novel computer architectures are addressed.

1 Introduction

Controversy often surrounds the reporting of results from scientific experimentation. Subtle points with unrecognized but profound effects are sometimes overlooked, and testing procedures are sometimes inadequate. Guidelines [CroDM79] for the reporting of the results of experimentation have helped to reduce some common errors and to clarify some issues for all concerned. As experience grows in new areas, such as the reporting of the results of computational experiments, it is natural to attempt to update and revise the guidelines to stimulate further discussion and to improve the reporting of results. In this spirit, the Council of the Mathematical Programming Society asked the Committee on Algorithms (COAL) to reconsider the existing guidelines and to offer extensions and modifications as deemed necessary. COAL appointed the authors to a subcommittee to work with the editor of *Mathematical Programming* and the Chairman of the Publications Committee to produce such a report.

The proliferation of powerful desktop workstations, minicomputers, mainframes, supercomputers, and novel machines with some form of parallel architecture, will continue to place increasing pressure on developers of algorithms to support their ideas with the results of computational experiments. Our hope is that we can supply a document that authors, editors, and referees can agree upon, and one that can provide a solid basis for the presentation of results.

In reviewing the existing guidelines, we found that much of what is stated is as valid today as when it was written, and is still worthwhile reading for anyone interested in the subject. Our main goal, therefore, is to supply some clarification on certain points and to address some new issues. Specifically, we believe that more discussion of proprietary codes is needed, and that some consideration of emerging novel architectures should be given. In addition we set forth in the next section the principles that we see as dominant. Our report is not meant as a primer on computational testing. There are several reasons for this, principally that it was not the charge of this committee. More importantly, perhaps, is that this is an enormous task requiring discussions of basic concepts in statistics, experiment design, numerical analysis, and software engineering. While we think that this is an important undertaking and would encourage someone to do it, it is beyond the scope of our effort.

As noted, this report is not a new set of guidelines. We have two goals: (1) to present a concise set of principles to guide authors, editors and referees in assessing computational tests, and (2) to clarify existing guidelines in certain areas. We intend for it to be an addition to the literature on computational experimentation and to be attached to the earlier report [CroDM79]. We recommend that editors routinely communicate the three principles below to referees, and that these principles become part of the editorial policy of the journal.

1.1 Principles

After thinking about rules for conducting and reporting computational tests, we believe that it is impossible to write a comprehensive list. Instead we settled on three principles that apply in all cases. These are the same principles that govern all experimental sciences: in papers that contain claims about computational performance,

REFERENCES

[Bus80] Bus, J. C. P., "Performance Measures for Evaluating Mathematical Programming Software", COAL Newsletter, January 1980, p. 8.

[CroDM79] Crowder, H. P., R. S. Dembo, and J. M. Mulvey, "On Reporting Computational Experiments with Mathematical Software", ACM Trans. on Math. Soft., 5, 1979, pp. 193-203.

[CroS80] Crowder, H. P. and P. B. Saunders, "Results of a Survey on MP Performance Indicators," COAL Newsletter, January 1980, pp. 2-6.

[DonBMS79] Dongarra, J. J., J. R. Bunch, C. B. Moler, G. W. Stewart, *Linpack Users' Guide*, SIAM Philadelphia, 1979.

[Gay83] Gay, D. M., "Algorithm 611. Subroutines for Unconstrained Minimization Using a Model/Trust-Region Approach", ACM Trans. Math. Soft., 9, (1983), pp. 503-524.

[Gay85] Gay, D. M., "Electronic Mail Distribution of Linear Programming Test Problems." Mathematical Programming Society COAL Newsletter 13, December 1985.

[GilHJLSS77] Gilsinn, J., K. Hoffman, R. H. F. Jackson, E. Leyendecker, P. Saunders, and D. Shier, "Methodology and Analysis for Comparing Discrete Linear L1 Approximation Codes", Communications in Statistics, 136, 1977, pp. 399-413.

[Hie81] Hiebert, K. L., "An Evaluation of Mathematical Software that Solves Nonlinear Least Squares Problems", ACM Trans. Math. Soft., 7, 1981, pp. 1-16.

[Hie82] Hiebert, K. L., "An Evaluation of Mathematical Software that Solves Systems of Nonlinear Equations", ACM Trans. Math. Soft., 8, 1982, pp. 5-20.

[HoaKP82] Hoaglin, D. C., V. C. Klema, and S. C. Peters, "Exploratory Data Analysis in a Study of the Performance of Nonlinear Optimization Routines", ACM Trans. Math. Soft., 8, 1982, pp. 145-162.

[HocS81] Hoch. W. and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*. Lecture Notes in Economics and Mathematical Systems No. 187, Springer-Verlag, New York, 1981.

[HocJ81] Hockney, R. and C. Jesshope, *Parallel Computers: Architecture, Programming and Algorithms*, Adam Hilger, Bristol, 1981.

[HofMSW53] Hoffman, A., M. Mannos, D. Sakolowsky, and N. Wieymann, "Computational Experience in Solving Linear Programs", J. SIAM, 1, 1953, pp. 17-33.

[HofJ82] Hoffman, K. L. and R. H. F. Jackson, "In Pursuit of a Methodology for Testing Mathematical Programming Software", in J. M. Mulvey, ed., *Evaluating Mathematical Programming Techniques, Lecture Notes in Economics and Mathematical Systems*, 199, Springer-Verlag, New York, 1982, pp. 177-199.

[JacM78] Jackson, R. H. F. and J. M. Mulvey, "A Critical Review of Comparisons of Mathematical Programming Algorithms and Software (1953-1977), J. Research of the National Bureau of Standards, 83, November 1978.

[Lyn79] Lyness, J. N., "Performance Profiles and Software Evaluation," Argonne National Laboratories TM 343, 1979.

[MieG78] Miele, A., S. Gonzales, "On the Comparative Evaluation of Algorithms for Mathematical Programming Problems", Nonlinear Programming, 3, New York, 1978, pp. 337-359.

[MorGH81] More, J. J., B. S. Garbow, and K. E. Hillstrom, "Testing Unconstrained Optimization Software", ACM Trans. Math. Soft., 7, 1981, pp. 17-41.

[Par87] Pardalos, P. M., "Generation of Large-Scale Quadratic Programs for Use as Global Optimization Test Problems", ACM Trans. Math. Soft., 13, 1987, pp. 133-137.

DRAFT

Report of the Ad Hoc Committee to Revise the Guidelines for Reporting Computational Experiments in Mathematical Programming

Richard H.F. Jackson, Chair[1] Paul T. Boggs[1] Stephen G. Nash[2]
Susan Powell[3]

[1]Center for Applied Mathematics, National Institute for Standards and Technology, Gaithersburg, MD, U.S.A.

[2]Operations Research and Applied Statistics Department, George Mason University, Fairfax, VA, U.S.A.

[3]London School of Economics & Political Science, London, U.K.

NOTES FROM THE (retiring) U.S. CO-EDITOR

I would like to echo the observations of Jens Clausen that the COAL Newsletter can play a major role in this exciting period in which many researchers in mathematical programming are enthusiastically tackling practical problems of hundreds of thousands and sometimes millions of variables. Our ability to formulate and efficiently solve such enormous problems offers the opportunity for increased prominence in science, engineering, and commerce, and poses new challenges in terms of combining optimization theory with computer software and hardware research. These issues are also considered in the draft report (on the following pages) of the committee established to revise the guidelines for reporting computational experiments in mathematical programming. Readers with suggestions for substantive changes or additions to the draft report may submit them in writing to one of the authors of the draft or to COAL officers.

Robert R. Meyer

CALENDAR

1989

July 23-26

Osaka, Japan: TIMS XXIX/Osaka '89.
Information: TIMS XXIX, TIMS, 290 Westminster Street., Providence, RI 02903, U.S.A.

August 29-31

Berlin, GDR: IFAC Symposium on Large Scale Systems: Theory and Applications.
Information: Prof. Dr. Dietrich Ohse, Fachbereich Wirtschaftswissenschaften, Johann Wolfgang Goethe Univ., Postfach 111932, D-6000 Frankfurt/Main, West Germany.

October 16-18

New York: ORSA/TIMS National Meeting.
Information: ORSA/TIMS NY, 1800 Diagonal Road, Suite 220, Alexandria, VA 22314, U.S.A.

December 11-13

Chicago: Fourth SIAM Conference on Parallel Processing for Scientific Computation.
Information: SIAM, 14th Floor, 117 S. 17th Street, Philadelphia, PA 19103, U.S.A.

1990

June 25-29

Athens: 12th Triennial Conference on Operations Research (IFORS '90).
Information: IFORS XII Secretariat, Hellanic Operational Research Society, 20 Solomon str., GR-10682, Athens, Greece.

COAL OBJECTIVES

The Committee on Algorithms is involved in computational developments in mathematical programming. There are three major goals: (1) ensuring a suitable basis for comparing algorithms, (2) acting as a focal point for computer programs that are available for general calculations and for test problems, and (3) encouraging those who distribute programs to meet certain standards of portability, testing, ease of use and documentation.

NEWSLETTER OBJECTIVES

The newsletter's primary objective is to provide a vehicle for the rapid dissemination of new results in computational mathematical programming. To date, our profession has not developed a clear understanding of the issues of how computational tests should be carried out, how the results of these tests should be presented in the literature, or how mathematical programming algorithms should be properly evaluated and compared. These issues will be addressed in the newsletter.

NOTES FROM THE EUROPEAN CO-EDITOR

At the 13th MPS Symposium in Tokyo, the editorial board of the COAL Newsletter was changed in the "standard way": The co-editor not in charge of production and distribution was asked to take over these obligations, a new co-editor was appointed and the immediate past co-editor was elected as chairman of COAL. The results of this process in terms of persons are that the U.S. co-editor from 1989 is Faiz A. Al-Khayyal of the School of Industrial & Systems Engineering, Georgia Institute of Technology, that I take over production and distribution, and that Bob Meyer leaves the editorial board. I take the opportunity to thank Bob for a good job as past co-editor. On several occasions it has been difficult to get a sufficient amount of material for the Newsletter, and Bob has made large efforts to persuade people to contribute.

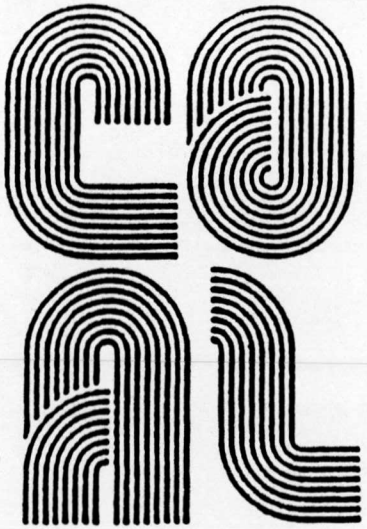
Following up on this issue, I would like to encourage the readers of the Newsletter to actually use this as "a vehicle for rapid dissemination of new results in computational mathematical programming". Traditionally the time passing from a paper is written until it is published by one of the regular journals is 1-2 years. COAL encourages authors to produce short versions of their papers containing the information necessary to understand the results and relate these to existing literature. The publication of this type of papers would indeed make the Newsletter fulfill its objectives.

The activities on the 13th MPS Symposium suggest that from the viewpoint of computational mathematical programming, the coming years will be particularly exciting in the fields of interior point methods for LP and IP problems, parallel computation in mathematical programming and guidelines for computational testing. I hope the Newsletter will be able to contribute to the discussions through the material provided by its readership.

Jens Clausen

COMMITTEE ON ALGORITHMS OF THE MATHEMATICAL PROGRAMMING SOCIETY

ROBERT R. MEYER
UNIVERSITY OF WISCONSIN-MADISON
COMPUTER SCIENCES DEPARTMENT
1210 WEST DAYTON STREET
MADISON, WISCONSIN 53706



Mathematical Programming Society

Committee on Algorithms Newsletter

NO. 18
March 1989

JENS CLAUSEN
ROBERT R. MEYER
EDITORS

CONTENTS