4. H. J. Greenberg (1978), Design and Implementation of Optimization Software. Sijthoff and Noordhoff, Alphen aan de Rijn, Netherlands.

5. L. S. Lasdon, A. D. Waren, M. W. Ratner, A. Jain (1975), GRG System Documentation (Technical Memorandum CIS-75-01), and GRG User's Guide (Technical Memorandum CIS-75-02), Cleveland State University, Cleveland, Ohio, USA.

6. F. A. Lootsma (1972), The ALGOL 60 procedure minifun for solving non-linear optimization problems. Report 4761, Philips Research Laboratories, Eindhoven, Netherlands. Published in H. J. Greenberg (1978), pp. 397-445.

7. F. A. Lootsma (1974), Convergence Rates of Quadratic Exterior Penalty-Function Methods for Solving Constrained-Minimization Problems. Philips Res. Repts. 29, 1-12.

8. F. A. Lootsma (1979), Performance Evaluation of Non-linear Programming Codes from the Viewpoint of a Decision Maker. In L. D. Fosdick (ed.) Performance Evaluation of Numerical Software. North-Holland, Amsterdam, 1979, pp. 285-297.

9. Th. L. Saaty (1977), A Scaling Method for Priorities in Hierarchical Structures. J. Math. Psych. 15, 234-281.

10. E. Sandgren (1977), The Utility of Nonlinear Programming Algorithms. Thesis, Purdue University, Dept. of Mechanical Engineering, West Lafayette, INdiana 47907, USA.

11. K. Schittkowski (1978), Randomly Generated NLP Test Problems with Predetermined Solutions. Preprint 35, Math. Institut der Julius-Maximilians-Universität, Würzburg, Germany.

12. R. L. Staha (1973), Constrained Optimization via Moving Exterior Truncations. Thesis, The University of Texas at Austin, Texas 78712, USA.

13. R. L. Staha and D. M. Himmelblau (1973), Evaluation of Constrained Nonlinear Programming Techniques. Report, The University of Texas at Austin, Texas 78712, USA.

LETTERS TO THE EDITOR

Allow me to make some constructive remarks concerning the paper about performance evaluation of nonlinear programming codes by Freerk Lootsma.

First, I do not agree completely with your definitions of the performance criteria. Basically, I prefer to use only those criteria for a comparison of NLP software which could be evaluated numerically. All other non-measurable criteria should be interpreted as additional information about the implementation, usage, and domain of applications of an NLP program. Specifically, I would propose the following three measurable criteria:

(a) Efficiency related to accuracy
(b) Global convergence
(c) Reliability or robustness.

Their evaluation is described in my reports. I would not accept the idea that accuracy be included as a part of reliability, but rather that efficiency be related to the reached accuracy. In addition, it is not practicable to require the same termination rules for all optimization programs. The introduction of "global convergence" seems for me to be very important since in practical applications, the user is mainly interested in global solutions. All comparative tests performed so far showed that there is indeed a significant difference in global convergence between optimization programs.

Simplicity of use is not a performance indicator I can use in evaluation. There are several reasons. First, there is a fundamental difference in the design of an optimization program between those included in commercial subroutine libraries and those classified as "experimental codes." In the latter case, there are, for example, many more parameters under the control of the user which influence the internal computations. In general, such codes are more difficult to evaluate. Furthermore, simplicity of use depends on the problem type. If, for example, lower and upper bounds or linear constraints are handled separately by the program, then it is easier to implement problems with bounds or linear constraints in this case, but more difficult to implement problems without bounds or linear constraints. Finally, many other properties could influence the simplicity of use. To give an example, consider Powell's VF02AD of the Harwell Subroutine Library. If the library is available, then it is very easy to implement the 250 additional statements of VF02AD. But if HSL is not available, the user has to do some work to implement VF02AD, especially to develop a special subprogram for inner products in machine language.

Program organization is an important feature for the implementation of NLP codes. But I cannot recommend any kind of measurement in the sense of "good" or "bad" organized programs. This subject should contain technical details and information about sources (availability, part of a library or not, price), implementation (programming language, precision, length, array organization, modularity), documentation (including input and output facilities), domain of applications (numerical differentiation, bounds or linear constraints included), usage (sensitivity to input parameters, provision of problem functions and gradients) and mathematical background.

LETTERS TO THE EDITOR

All these non-quantifiable code descriptors have the purpose of providing the decision maker with sufficient information to determine a subset of all considered programs which could solve his problems and satisfy his implementation and usage requirements (for example all codes up to 1500 statements, computation of objective function and constraints in one subroutine). The final decision within this subset should depend then on the measurable performance criteria described above.

The priority theory of Saaty is a useful tool to allow an evaluation of performance criteria which could not be calculated numerically. But I see some drawbacks. As described above, I would recommend that measurable criteria be used to compare optimization programs. However, even if one proceeds from the performance criteria described in the Lootsma paper, I find that most of the criteria listed consist of several items with different weights (for example efficiency consists of CPU time, number of function and gradient calls). This requires a more detailed performance evaluation since first you must apply the theory to determine the weights and the items of each single criterion.

My remarks should be considered as a part of the discussion about performance evaluation of NLP software. I am not even sure if all my statements will remain unchanged after a first final balance, but I am convinced that we have to do a lot of experimental work to get an impression on how optimization programs should be tested.

Klaus Schittkowski
Der Universitat Wurzburg

****

During one of the COAL sessions in Montreal and summarized in this Newsletter, F. Lootsma proposed a multi-criteria approach to evaluating optimization algorithms and software based on Saaty's priority theory. While agreeing in principal, I cannot see any justification for limiting oneself to priority theory. In fact, there are many fine multi-criteria techniques and the interested should see the following references for further information:

1. Dyer, J. S., "Interactive Goal Programming," Management Science, 19, 1972, pp.62-70.

2. Keeney, R. and Raiffa, H., Decision Making with Multiple Objectives: Value and Preference Tradeoffs, John Wiley, 1976.

3. Kornbluth, J. S. H., "A Survey of Goal Programming," Omega, 1, 1973, pp.193-205.

4. Zionts, S. and Wallenius, J., "An Interactive Programming Method for Solving the Multiple Criteria Problem," Management Science, 22, 1976, pp.652-663.

John M. Mulvey
Princeton University

As in many decision problems, there is a general vagueness in the significance of the decision criteria, and the calculated priorities should accordingly be used as a guideline for the deliberations. Obviously, one is only interested in orders of magnitude, not in accurate values of the priorities. For performance evaluation of codes, this implies that rough estimates of the relative performance must be established under various performance criteria.

There is no ideal code for non-linear programming problems. This was already known, but Table 2 shows how easily the preference for a GRG code may disappear if the priorities of the performance criteria are modified. Obviously, there is a 'market' for a GRG code and for the COMET code. Hence, it is not the task of performance evaluation to designate the best code in a variety of circumstances. It only has the humble task to provide the material for the selection of codes in various cases.

TABLE 2

Final Score of Codes GRG, COMET, and PENF. The columns 1-3 contain the priorities of the codes under each performance criterion separately, the columns 5-7 the same priorities multiplied by the priority (column 4) of the criterion concerned.

	Factor	GRG	COMET	PENF	priority	GRG	COMET	PENF
F_1	Robustness	0.43	0.43	0.14	0.40	0.17	0.17	0.06
F_2	Efficiency	0.54	0.30	0.16	0.20	0.11	0.06	0.03
F_3	Capacity	0.33	0.33	0.33	0.10	0.03	0.03	0.03
F_4	Simplicity of Use	0.50	0.25	0.25	0.15	0.08	0.04	0.04
F_5	Program Org.	0.07	0.48	0.45	0.15	0.01	0.07	0.07
	Final Score					0.40	0.37	0.23

References

1. J. Abadie (1975), Méthode du gradient réduit généralisé: le code GRGA. Electricite de France, Service I.M.A., Note HI/1756/00, Paris.

2. A. R. Colville (1968), A comparative study of non-linear programming codes, IBM New York Scientific Center, Technical Report 320-2949.

3. A. V. Fiacco and G. P. McCormick (1968), Non-linear Programming: Sequential Unconstrained Minimization Techniques. Wiley, New York.

Capacity. The comparative studies do not supply sufficient information to distinguish the codes from each other. Hence, equal priorities 0.33 are assigned to each of the codes.

Simplicity of Use. The comparative studies do not pay much attention to this criterion, but since GRG codes are reported to be less sensitive to parameters which control the computational process, the priorities 0.50, 0.25, and 0.25 are assigned to the respective codes.

Program Organization. This criterion is also neglected in the comparative studies, but the additional information that the GRG codes take more than 4000 lines of coding, the COMET code about 600 lines, and PENF codes somewhat more than 600 lines, leads to the assignment below.

Program Organization (factor F_5)

	GRG	COMET	PENF	
GRG	1	1/7	1/6	0.07
COMET	7	1	1	0.49
PENF	6	1	1	0.44

After some adjustments, the priorities are set to 0.07, 0.48, and 0.45 respectively.

The final results are displayed in Table 2 where we find the priorities of the codes for each factor, separately (columns 1-3) and multiplied by the priority (column 4) of the performance criterion concerned (columns 5-7), as well as the final score per code. We do not expect that the specialist will immediately obey the numerical results and choose a GRG code. Table 2, however, can be used for a structured analysis of the situation and for a structured preparation of the decision. It clearly demonstrates the impact of the performance criteria as well as the strengths and weaknesses of the codes, thus guiding the deliberations toward the decisive arguments. This is typically the objective of priority theory.

4. Concluding Remarks

The above example clearly shows what performance evaluation can do, and what it cannot do. The comparative studies have been used to assign priorities to the codes under various performance criteria, but it is up to a decision maker to identify and to weigh the relevant performance criteria. Priority theory provides the framework for an integrated assessment of codes.

ON THE DESIGN OF OPTIMIZATION SOFTWARE*

Jorge J. Moré, Argonne National Laboratory
9700 South Cass Avenue, Argonne, IL 60439

Summary

MINPACK is a research project whose long term goal is the development of a systematized collection of quality optimization software. One of the results of this project is a package, MINPACK-1, for the solution of systems of nonlinear equations and nonlinear least squares problems. This paper provides an outline of this package and of some of the design decisions made during the production of this package.

The goal of MINPACK-1 is to minimize the amount of effort required of the user and the computer to solve a particular problem. This goal demands that close attention be paid to the ease of use, reliability, and efficiency of MINPACK-1. Ease of use requires the careful design of the user documentation and user interface, while reliability and efficiency require that the algorithms have acceptable global and local convergence properties and that the implementations extend the domain of the algorithms as much as possible.

This paper does not provide a complete description of the design principles behind MINPACK-1, but rather illustrates some of these principles by considering specific examples. The concepts discussed in this paper are robustness, scale invariance, interface routines, and reverse communication.

The concept of robustness has been used with great success in other areas of mathematical software, but it is frequently given too little attention in optimization software. We illustrate the importance of this concept by discussing the robustness of a very simple but important calculation: cubic interpolation. The importance of this calculation derives from its use as a basic step in many one-dimensional optimization routines.

Similarly, although almost every successful developer of optimization algorithms is aware of the importance of scale invariance, the use of this concept in the implementation of optimization software has been overlooked by many. In our discussion of scale invariance, we note a connection between robustness and scale invariance and show how scale invariance can be used to decide between different versions of an algorithm.

Interface routines and reverse communication are concepts that have been used to facilitate the use of MINPACK-1. Some of the implications of these concepts are treated by discussing the design of an interface routine for a nonlinear least squares algorithm, and the implementation of an algorithm for checking that the user-supplied derivative information is consistent with the function values.

*Work performed at Cambridge University under the auspices of the U. S. Department of Energy and with partial support from the Science Research Council.

A SIMULATION TEST APPROACH TO THE EVALUATION OF NONLINEAR OPTIMIZATION ALGORITHMS

Kenneth E. Hillstrom, Argonne National Laboratory
9700 South Cass Avenue, Argonne, IL 60439

1. Introduction and Background

A simulation test technique has been developed to evaluate and compare unconstrained nonlinear optimization computer algorithms.

The simulation test technique is motivated by and addresses the need to recommend algorithms on the basis of overall performance to the user community, as opposed to fulfilling the requirements of algorithm developers for in-depth concentrated studies designed to isolate algorithm strengths and weaknesses.

In light of the above requirements, it became apparent that tests commonly used to evaluate and compare optimization algorithms are limited in scope and have the following characteristics: (1) a function, representative of topography or complexity encountered in applications, such as Rosenbrock's descending parabolic valley, is selected as a test problem; (2) a fixed initial estimate of the solution is used in all tests; and (3) the computational effort required to find the extrema is measured in terms of iterations or equivalent function evaluations. The tests commonly used are inadequate for at least two reasons. First, the tests were originally designed to probe for weaknesses in algorithms. In particular, tests using functions exhibiting difficult topographies encountered in real-world situations, such as curving valleys, saddle points, etc., were given initial approximations to the solution designed to test the algorithm by directing the search through the topography. For example, the starting point in Rosenbrock's problem initiates a path down the valley. However, when these tests are used to recommend algorithms to the user community for general usage, serious questions about the impartiality and reality are raised.

Second, computational effort measured in terms of iterations or equivalent function evaluations is subject to large discrepancies when comparing algorithms. Iterations generally vary greatly in scope from algorithm to algorithm. Equivalent function evaluation assignment may also lead to erroneous conclusions. As an example, the assignment of weights of 1 and N, respectively, to an N-variable function and gradient computation, when comparing derivative with derivative-free routines, may be unrealistic since the gradient computation is often less extensive than that of the function. Third, reliance on iterations or equivalent function evaluations alone as a measure ignores the often considerable overhead computational effort required in obtaining a solution.

The technique described here expands the scope of examination. The first limitation is addressed in part by utilizing a series of randomly generated starting points for each optimization problem instead of only one fixed point. Thus the initial estimates simulate those encountered in practice, and a survey is obtained of the algorithm's ability to converge along various paths. Moreover, if an algorithm has been tailored to solve a particular problem with a particular starting point, the random starts will aid in revealing its limited applicability. The second limitation is addressed by utilizing

It is easy to understand that robustness should have the highest priority, followed by efficiency, and we are convinced that simplicity of use and program organization cannot be neglected in nonlinear programming. It is not our intention, however, to state that the above priorities are valid in a majority of the research and development laboratories. We have only presented an example to illustrate decision making and the role of performance evaluation in the decision process. It is interesting to point at a recent questionnaire of the Committee on Algorithms: the answers show that high-quality documentation (here included in the criterion of simplicity of use) has top priority for many optimization specialists.

3. Priorities of Codes

In the above named comparative studies, several reduced-gradient and penalty-function codes have been tested, and the studies reveal that there is a considerable amount of consistency in the results. We do not therefore concentrate on a particular code in these categories, but we consider a generalized reduced-gradient code GRG and a penalty-function code PENF without further specifications. The COMET code of Staha [1973] for constrained optimization via moving exterior truncations is the only code in its category, however. We now consider the relative performance of the codes GRG, COMET, and PENF under each performance criterion separately.

Robustness. The studies of Staha [1973] and Sandgren [1977] show that GRG and COMET are equally robust: they solve practically the same number of test problems. For a PENF code, the number of unsolved test problems is roughly three times higher. This is expressed in the below matrix of relative significance

Robustness (factor F_1)

	GRG	COMET	PENF	
GRG	1	1	3	0.43
COMET	1	1	3	0.43
PENF	1/3	1/3	1	0.14
	0.43	0.43	0.14	1.00

resulting into priorities 0.43, 0.43, and 0.14 assigned to GRG, COMET, and PENF respectively.

Efficiency. The GRG codes appear to be roughly three times faster than PENF codes. In Staha [1973], the COMET code is competitive with GRG if derivatives are supplied by the user, but much slower with numerical differentation; the observation is confirmed by Sandgren [1977]. This is expressed in a matrix of relative significance, leading to the respective priorities 0.54, 0.30, and 0.16.

Table 1.

Matrix R of Relative Significance, and Approximations to Priorities of Performance Criteria by Normalized Row Sums, Normalized Inverted Column Sums, and Eigenvector corresponding to $\lambda_{max} = 5.18$.

FAKTOR	DESCRIPTION	F_1	F_2	F_3	F_4	F_5	ROW SUMS	NORMALIZED	EIGENVECTOR
F_1	ROBUSTNESS	1	3	3	2	4	13.00	0.40	0.42
F_2	EFFICIENCY	1/3	1	2	2	2	7.33	0.22	0.21
F_3	CAPACITY	1/3	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	2.83	0.09	0.09
F_4	SIMPLICITY OF USE	$\frac{1}{2}$	$\frac{1}{2}$	2	1	1	5.00	0.15	0.15
F_5	PROGRAM ORGANIZATION	$\frac{1}{4}$	$\frac{1}{2}$	2	1	1	4.75	0.14	0.13
	COLUMN SUMS	2.42	5.50	10.00	6.50	8.50			
	INVERTED NORMALIZED	0.43	0.19	0.10	0.16	0.12			

measures of both direct and indirect computational effort. The direct computational effort (DCE), i.e., the effort expended in the user-supplied routines, is measured in units of direct computational effort (DCU), which affords a medium of comparison between various types of optimization routines.

The indirect or overhead computational effort is measured in terms of DCU units per unit of elapsed time. A large number of DCUs per unit of time indicates relatively low overhead effort, while a small number indicates dominating overhead.

2. The Simulation Test Technique

The test technique simulates problems optimization algorithms encounter in practice by employing a repertoire of problems representing various topographies (descending curved valleys, saddle points, ridges, etc.), dimensions, degress of nonlinearity (e.g., linear to exponential), and minima, addressing them from various randomly generated initial approximations to the solution and recording their performances in the form of statistical summaries. These summaries, consisting of categorized results and statistical averages, are generated for each algorithm as tested over members of the problem set. The individual tests are composed of a series of runs from random starts over a member of the problem set.

****

ON TESTING ALGORITHMS FOR MATHEMATICAL PROGRAMMING PROBLEMS

A. Miele, S. Gonzalez, and A. K. Wu, Rice University
Aero-Astronautics Group, 230 Ryon Bldg., Houston, TX 77001

Abstract

This paper considers the comparative evaluation of algorithms for mathematical programming problems. It is concerned with the measurement of computational speed and examines critically the concept of equivalent number of function evaluations N_e . Does this quantity constitute a fair way of comparing different algorithms?

The answer to the above question depends strongly on whether or not analytical expressions for the components of the gradient and the elements of the Hessian matrix are available. It also depends on the relative importance of the computational effort associated with algorithmic operations vis-a-vis the computational effort associated with function evaluations.

Both theoretical considerations and extensive numerical examples carried out in conjunction with the Fletcher-Reeves algorithm, the Davidon-Fletcher-Powell algorithms, and the quasilinearization algorithms suggest the following: the N_e concept, while accurate in some cases, has drawbacks in other cases; indeed, it might lead to a distorted view of the relative importance of an algorithm with respect to another.

The above distortion can be corrected through the introduction of a more general parameter, the time-equivalent number of function evaluations $N_e = T/\tau_0$, where T denotes the CPU time required to solve a particular problem on a particular computer and τ_0 denotes the CPU time required to evaluate the objective function once on that computer. This generalized parameter is constructed so as to reflect accurately the computational effort associated with function evaluations and algorithmic operations.

From the analyses performed and the results obtained, it is inferred that, due to the weaknesses of the N_e concept, the use of the N_e concept is advisable. In effect, this is the same as stating that, in spite of its obvious shortcomings, the direct measurement of the CPU time is still the more reliable way of comparing different minimization algorithms.

A condensed version of this paper, titled On the Comparative Evaluation of Algorithms for Mathematical Programming Problems, has appeared in the book Nonlinear Programming 3, edited by O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, Academic Press, New York, pp. 337-359, 1978. Copies of the above report and paper can be obtained by writing to: Dr. A. Miele, Aero-Astronautics Group, 230 Ryon Building, Rice University, Houston, Texas, 77001.

****

A COMPARATIVE STUDY OF METHODS FOR IDENTIFYING REDUNDANT CONSTRAINTS IN LINEAR PROGRAMMING

Mark Karwan, Jan Telgen, and Stanley Zionts
State University of New York at Buffalo
Dept. of Mgt. Sci. & Systems, Buffalo, NY 14214

We are in the process of coordinating a study on redundancy in mathematical programming together with approximately 15 researchers from all over the world. The project involves the collection, programming, and testing of about ten methods for identifying redundant constraints in mathematical (mainly linear) programming. Our study includes preparing a typology of methods, defining appropriate terms and programming and testing the methods. The result will be a comprehensive volume on the state-of-the-art of the methods. The purpose of this note is to seek input on our plans and procedures. We are planning to program the methods in FORTRAN using common subroutines to the extent possible. Where the methods involve simplex iterations, we plan to have both a full simplex tableau routine for sufficiently small problems, as well as a product-form routine. We plan to use randomly generated problems for testing as well as practical problems. For the random problems, we plan to use an adaptation of LPGENR [1] to generate problems of the following form:

$$\begin{array}{ll} \text{Maximize} & c'x \\ \text{subject to:} & Ax \leq b \\ & x \geq 0 . \end{array}$$

a) Domain of applications: the type or class of problems for which the code has been designed.

b) Robustness (reliability and accuracy): the power to solve problems in the domain of application with the required accuracy.

c) Efficiency: the effort (usually measured in terms of equivalent function evaluations or CPU time) necessary to solve problems in the domain of application with the required accuracy.

d) Capacity: the maximum size of the problems that can generally be solved by the code.

e) Simplicity of use: the presence of high-quality documentation and user-oriented features (output provisions, numerical differentiation); conceptual simplicity of the underlying algorithm.

f) Program organization: the language, the length, and the structure of the code.

Basically, reduced gradients, penalty functions, and moving truncations have the same domain of applications. Hence, we shall only be concerned with the remaining performance criteria (the factors, to be designated by the symbols $F_1, F_2, \ldots, F_5$). For each pair of factors, the relative significance may be set to one of the values recommended by Saaty [1977]. Thus,

$r_{ij} = 1$, if F_i and F_j are felt to be equally important;

$r_{ij} = 3$, if F_i is felt to be somewhat more important than F_j;

$r_{ij} = 5$, if F_i is felt to be much more important than F_j.

The intermediate values of 2 and 4 are to be assigned in cases of doubt between two adjacent values. In certain cases, values higher than 5 can also be assigned. The matrix R with elements r_{ij} is reciprocal, since we set $r_{ji} = 1/r_{ij}$.

The results in the hypothetical decision situation are displayed in Table 1. The calculated approximations to the priorities are also shown in that table. We assume that after some deliberations the priorities would be adjusted as follows:

F_1 (robustness) 0.40

F_2 (efficiency) 0.20

F_3 (capacity) 0.10

F_4 (simplicity of use) 0.15

F_5 (program organization) 0.15.

PERFORMANCE EVALUATION OF NON-LINEAR PROGRAMMING CODES VIA MULTI-CRITERIA DECISION ANALYSIS

F. A. Lootsma, University of Technology, Delft, Netherlands

1. Introduction

This paper, a brief version of an earlier publication (Lootsma [1979]), shows how a method for multi-criteria decision analysis can be used for an integrated appreciation of non-linear programming codes, whereby both the performance criteria of a decision maker and the results of some recent comparative studies (Colville [1968], Staha and Himmelblau [1973], Sandgren [1977], Schittkowski [1978]) are taken into account. We sketch the role of performance evaluation from the viewpoint of an imaginary optimization specialist who will be responsible for non-linear optimization in an industrial research and development laboratory. He knows that there are various strategies to solve non-linear programming problems, but since it is out of the question that he should have the time and energy to become master of all of them, he decides to select one particular strategy. In order to make the selection, he also decides to use the above-named comparative studies. In these studies, reduced-gradient methods (see Abadie [1975], Lasdon et al. [1975]) and penalty-function techniques (see Fiacco and McCormick [1968], Lootsma [1972]) were heavily tested; in two studies (Staha and Himmelblau [1973] and Sandgren [1977]), remarkable properties were reported with respect to moving exterior truncations (see Lootsma [1974]). Other strategies, however, received less attention so that they will here be left out of considerations.

As a vehicle for discussion we use the priority theory of Saaty [1977] which has been developed to weigh the significant factors (decision criteria, performance criteria) in a decision problem via pairwise comparison of the factors. Each ratio expressing the relative significance of a pair of factors is displayed in a matrix. Finally, the weights (the so-called priorities) of the factors are obtained by an eigenvalue analysis. In practice, the normalized row sums and the normalized, inverted column sums of the matrix in question provide workable approximations to the eigenvector of estimated priorities.

In fact, the optimization specialist runs up against a two-level decision problem. First, he has to identify and to weigh the performance criteria which are relevant in the given situation. Second, he has to compare some codes, and he has to establish their relative performance under each of the criteria separately. Finally, adding the weights (priorities) per code multiplied by the weights of the criteria, he obtains a score for each code. The highest score designates the code to be selected.

2. Performance criteria

The following list of performance criteria for non-linear programming codes is not exhaustive, but we have the impression that it contains at least the criteria which are predominant.

The tentative design is as follows:

A. 10 methods

B. 5 repetitions per cell

C. 2 levels of degeneracy (no degeneracy versus 50% degeneracy at the starting solution and 50% degeneracy at the optimal solution)

D. 3 levels of problem size and planned redundancy as follows:

	Constraints	Structural Variables (excluding slack variables)	Redundancy
level 1	10	10	no planned redundancy
level 2	20	10	50% planned redundancy generated from level 1 problems by adding linear combinations of two constraints. This will permit pairwise comparisons with level 1.
level 3	20	10	no planned redundancy

The above involves 300 tests. As a result of the above experiment, as well as those with larger real problems, additional experiments will be undertaken.

We welcome comments and suggestions on the proposed design.

Reference

Layman, C. H. and R. P. O'Neill, "A Study of the Effect of LP Parameters on Algorithm Performance," in W. W. White, Computers and Mathematical Programming, National Bureau of Standards (1978).

IMPLEMENTATION AND TESTING OF OPTIMIZATION SOFTWARE*

Jorge J. Moré, Argonne National Laboratory
9700 South Cass Avenue, Argonne, IL 60439

Summary

This paper is concerned with optimization software which must perform satisfactorily on a wide range of machines and compilers. This requirement demands that the development of the software must adhere to strict standards; the purpose of performance testing is to verify that the implementation of an algorithm satisfies these standards.

During the development of optimization software at Argonne National Laboratory we have found that it is necessary to test software at two levels. The first level of testing is concerned with the robustness and reliability of the implementation, while the second level tests the overall performance of the algorithm.

A robust implementation extends the domain of the algorithm so that it copes with as many problems as possible without a serious loss of efficiency. A reliable implementation performs well-defined calculations accurately and efficiently. These definitions have important implications for optimization software and we discuss, in particular, the implications to failures of optimization algorithms due to destructive overflows and underflows, inadequate termination criteria, and invalid arguments. Since we are concerned with performance on a wide range of machines and compilers, we also outline the approach used at Argonne to develop transportable optimization software.

There are many approaches to performance testing, but most of these approaches restrict the test problems, and as a consequence do not really test the overall performance of the implementation. Moreover, performance in optimization is often equated with efficiency, and thus accuracy and reliability have not received sufficient attention. To address these needs, we suggest guidelines for the implementation of optimization software and show the (often dramatic) effect that these guidelines have on the performance of the implementation. In addition we discuss the weaknesses in the usual approaches to performance testing of optimization software and suggest some remedies. The discussion of performance is illustrated by testing a nonlinear least squares algorithm.

*Work performed at Cambridge University under the auspices of the U. S. Department of Energy, with partial support from the U. S. Army Research Office (contract DAA629-78-M-0152) and the Science Research Council.

robustness or efficiency (see e.g. HILLSTROM [1977], LYNESS [1979], GILSINN et al. [1977], SCHITTKOWSKI [1979]). None of these quantities define, by themselves, robustness or efficiency. However, collecting the values of such quantities for certain programs may give enough numerical evidence for program selection.

References

Gilsinn, J. et al. [1977], Methodology and analysis for comparing discrete linear L_1 approximation codes. Commun. Statis.-Simula. Computa., B6(4), 399-413.

Hillstrom, K. E., [1977], A simulation test approach to the evaluation of nonlinear optimization algorithms. ACM TOMS 3, 305-315.

Lyness, J. N., [1979], Performance profiles and software evaluation. In: L. D. Fosdick (ed.), Performance evaluation of numerical software, North Holland

Lyness, J. N. and C. Greenwell [1977], A pilot scheme for minimization software evaluation, Argonne Nat. Lab. report TM 323.

Moré, J. J. [1979], Implementation and testing of optimization software. In: L. D. Fosdick (ed.), Performance evaluation of numerical software, North Holland.

Ragsdell, K. M., [1978], On some experiments which delimit the utility of nonlinear programming algorithms, paper pres. at ORSA/TIMS National meeting (Nov. 1978).

Sandgren, E., [1977], The utility of nonlinear programming algorithms. Thesis, Purdue Univ. Dept. of Mechan. Eng.

Schittkowski, K., [1979], A numerical comparison of 13 nonlinear programming codes with randomly generated test problems. To appear in: Dixon, L.C.W. & G.P. Szego (eds.) Numerical optimization of dynamical systems. North Holland.

Therefore, CPU-time used by a program to solve a problem is a performance measure which is not portable to other computer environments. Thus, this performance measure is not useful. We should develop some efficiency measures which are more or less machine and language independent. A second observation to be made about efficiency is particularly important for nonlinear programming software. The time required to evaluate the user supplied software (objective function, gradients, constraints, etc.) in such programs may vary largely, even for problems with the same number of variables and constraints. Dependent on the overhead costs of a program, one program may be efficient for solving cheap problems and inefficient for solving expensive problems or vice versa. Therefore, at least for nonlinear programming, one should express efficiency measures as a function of the time required to evaluate user supplied software.

4. Performance measures

Performance measures that can be used for evaluating software are:

1. Documentation standards; one should be able to use software easily after reading the documentation.

2. Well-structuredness of programs; maintenance, error-tracking, and performing minor changes should be relatively easy.

3. Program portability; programming should be in standard language; arithmetical errors (overflow, underflow, logarithmic errors, etc.) should be avoided as much as possible; use of machine constants (machine precision, dwarf, giant, etc.) should be mentioned clearly.

4. Reliability; either a correct answer is produced or the program terminates with an understandable and useful error message; we say that an answer is correct if it meets the requirements claimed in the documentation.

5. Robustness; the program should recover gracefully from specific difficult situations; it should solve problems with peculiar properties as much as possible; it should solve as many problems as possible from the class to which it is designed.

6. Efficiency; some measure(s) which reflect the time and storage required to solve a problem.

Notice that the first three measures are particularly important for evaluation of implementations of a certain algorithm. These measures do not depend on the underlying algorithm.

The above enumeration, in fact, only gives a rough subdivision of possible performance measures without defining any quantities to give a precise quantification of the given notions. In fact, particularly for robustness and efficiency, one can imagine a lot of quantities all reflecting some aspects of

RANDOM GENERATION OF PROBLEMS WITH A GIVEN STRUCTURE

R. S. Dembo, Yale University
School of Organization & Mgt., Box 1A, New Haven, CT 60520

Ever since the advent of digital computers there appears to have been a controversy over whether one should compare algorithms using randomly generated problems or problems selected from real-world applications. The purpose of this note is not to take sides on this issue but rather to offer an alternative that is somewhat of a middle path.

The idea is simple:

(i) Select a mathematical programming model of a real-world system either because it solves a problem that is of wide interest, or because it poses difficulties for a class of algorithms.

(ii) Develop a random problem generator based on this model by perturbing parameters in a way that has physical meaning in the situation being modeled.

A problem generator created in this way then may be viewed as an attempt to simulate the conditions under which the real-world system operates. The importance of random generation in this context is that it enables one to compare the performance of competing algorithms, on this class of problems, using statistical techniques [1].

To give an example of such a generator, I have selected a well-documented optimal design problem that has been the subject of some interest in recent testing of algorithms for constrained nonlinear programming [4,5].

The problem concerns the optimal design of a multistage membrane separating process in which two gases are to be separated. A complete description of the physical characteristics of the process and a derivation of the nonlinear programming model are given in [2]. I will use the three stage process depicted in that report to demonstrate the idea.

An important parameter in the process is the degree of separation, X_R , which is a measure of the purity of the final product. In order to create a random problem generator that simulates the behavior of the system for different product purity requirements one would proceed as follows.

(i) Select a range of "interesting" X_R values; e.g. [.88, .98]

(ii) Select a distribution for X_R over this range; e.g. the uniform distribution

(iii) Draw values of X_R randomly from this distribution; each value of X_R corresponds to an NLP drawn randomly from the class of "optimal membrane separation design models."

A more interesting and comprehensive problem generator based on this principle is now under development at Yale. It is based on the water distribution network of the city of Dallas and will be reported on in a future issue.

References

[1] Dembo, R. S. and Mulvey, J., "On the Analysis and Comparison of Mathematical Programming Algorithms and Software," in W. W. White, ed., Computers and Mathematical Programming, National Bureau of Standards Special Publication 502 (1978).

[2] Dembo, R. S. and Avriel, M. "Optimal Design of a Membrane Separation Process Using Signomial Programming," Mathematical Programming, Vol. 15 (1978), pp.12-25.

[3] Dembo, R. S., "A Set of Geometric Programming Test Problems and Their Solutions," Mathematical Programming, Vol. 10 (1976), pp.192-312.

[4] Powell, M. J. D., "Constrained Optimization by a Variable Metric Method," Report DAMTP 77/NA6, University of Cambridge, Cambridge, England (1977).

[5] Minkoff, M., private communication.

****

NONSMOOTH OPTIMIZATION PROBLEMS

A Workshop on Nonsmooth Optimization was held at IIASA from March 28 to April 8, 1977. (Nonsmooth optimization problems are problems having functions for which gradients exist almost everywhere, but are not continuous.) In the proceedings of that workshop [1], four nonsmooth optimization problems are presented. The editors of this proceeding explain in the preface of that text that these problems "were selected because they are easy to work with and because they represent both the field of applicability and of the range of difficulty of NSO. Problems 1 and 3 are examples of minimax problems and are not very difficult. Problem 2 is a nonconvex problem coming from a well-known NLP test problem [a modification of the Shell Dual Problem], and problem 4 involves a piecewise-linear function. The last two are sufficiently difficult to slow down considerably the speed of any of the NSO methods we know of."

The Committee on Algorithms is pleased to add these problems to its collection of test problems and thanks Robert Mifflin for bringing them to our attention.

[1] Lemarechal, Claude and Mifflin, Robert (1978), Nonsmooth Optimization, IIASA Proceedings Series, V3. Proceedings of a IIASA Workshop, March 28-April 8, 1977. Pergamon Press Ltd., Oxford, England.

PERFORMANCE MEASURES FOR EVALUATING MATHEMATICAL PROGRAMMING SOFTWARE

Jacques C. P. Bus, Mathematical Center, Amsterdam

1. Software evaluation and software selection

In this contribution I would like to emphasize first the essential distinction that has to be made between software evaluation and software selection. Software evaluation is the process by which one obtains data about the performance of software. This process should be well-defined and based on mathematical and statistical methods. It may yield extensive information about software performance for some very specific problems (HILLSTROM [1977], LYNESS [1979], LYNESS & GREENWELL [1977], MORE [1979]) or global information about the performance of software for large sets of test problems (RAGSDELL [1978], SANDGREN [1977], SCHITTKOWSKI [1979]). On the other hand, software selection is a subjective action. Based on information which is made available by software evaluation and dependent on one's objectives, one may select a certain piece of software for use in a certain environment. Although such a decision process may be defined mathematically (see LOOTSMA [1979] or the summary in this newsletter), the premises basic to such a process are subjective (e.g. one wants to solve one difficult and time consuming nonlinear problem or a long series of easy and cheap nonlinear problems, etc.). Unfortunately, the distinction between software evaluation (collecting objective data about the software) on the one hand and software selection (deciding what should be used in a certain situation) on the other hand, is not clearly given in most large comparative studies of mathematical programming software.

2. Evaluation of algorithms and programs

In many comparative studies conclusions about algorithms are drawn based on information about the performance of programs implementing such algorithms. This might yield wrong conclusions. In fact, a good implementation of a poor algorithm might be more robust or efficient than a poor implementation of a good algorithm. When programs are evaluated as black boxes, it is very hard to draw conclusions about the underlying algorithms from the performance of the programs. Therefore, it might be useful to distinguish between evaluation of implementations of different algorithms. Moreover, one might use a different set of performance measures for both evaluations. Before discussing specific performance measures we shall give some comments on the use of CPU-time to measure efficiency of programs.

3. Use of CPU-time to measure efficiency

It is well-known that the CPU-time required by some program for solving some problems is highly dependent on

-- the computer
-- the programming language and compiler (options)
-- the computer environment (especially on multi-user systems)
-- the software library used (special built-in error mechanism, use of basic routines in machine language etc.).

PERFORMANCE MEASURES FOR EVALUATING MP SOFTWARE

	ACADEMIC		INDUSTRY		CONSULTING		GOVERNMENT		OTHER	
NO. OF RESPONSES	159		60		25		17		14	
	Rank	Score	Rank	Score	Rank	Score	Rank	Score	Rank	Score
CPU Time	2	1.24	2	1.15	1	1.44	----		----	
Documentation	1	1.25	5	.97	5	.64	1	1.52	1	1.5
Rel. Dif. Obj. Fn.	3	.98	4	1.00	3	.76	4	.77	----	
Cost/Run	4	.75	1	1.35	2	1.00	2	.88	----	
Sensitivity Measures	4	.75	3	1.10	4	.72	4	.77	2	1.1
No. Funct. Evals.	----		----		----		----		----	
Max. Rel. Error in Sol. Vec.	----		----		----		4	.77	----	
Ease of Modification	----		----		----		4	.77	3	1.0
Correct Digits--Obj. Fn.	----		----		----		3	.82	----	
System Design	----		----		----		4	.77	----	
Input Scheme	----		----		----		----		4	.93
Euclidean Dist. Sol.	----		----		----		----		4	.93

TABLE III
TOP FIVE PERFORMANCE INDICATORS FOR RESPONDENTS
CLASSIFIED BY CURRENT EMPLOYMENT

	DEV./CODED RESPONDENT		BUY FROM VENDOR		FREE FROM VENDOR		BUY FROM DEVELOPER		FREE FROM DEVELOPER	
NO. OF RESPONDENTS	231		77		63		78		124	
	Rank	Score	Rank	Score	Rank	Score	Rank	Score	Rank	Score
CPU Time	2	1.12	4	.90	2	1.25	3	.97	2	1.12
Documentation	1	1.18	1	1.35	1	1.76	1	1.14	1	1.30
Rel. Dif. Obj. Fn.	3	.99	5	.81	4	1.13	5	.75	3	.92
Cost/run	5	.81	2	1.34	3	1.14	4	.83	4	.76
Sensitivity Measures	4	.84	3	.91	5	.98	2	1.00	5	.74

TABLE IV
TOP FIVE PERFORMANCE INDICATORS FOR RESPONDENTS
CLASSIFIED BY USUAL SOURCE OF MP SOFTWARE

OUTGOING CHAIRMAN'S COMMENTS

The number of presentations involving computational tests of algorithms and software at four of the past five Mathematical Programming Symposia are shown below:

Princeton (67)	3	Budapest (76)	12
Stanford (73)	22	Montreal (79)	32

Clearly this area is growing and becoming more respectible. As further evidence, the COAL business meeting in Montreal alone attracted over 25 people from 10 countries at 5:00 p.m. after seven hours of meetings. Attendance at COAL sponsored sessions has been increasing and lively exchanges are commonplace. Likewise, the experiments themselves are considerably more complete than experiments conducted several years before.

These positive signs are counterbalanced in part by several factors:

(1) more questions are often unearthed during an empirical investigation than are answered,
(2) computational analysis is an extremely expensive activity that may not be generalizable outside the immediate set of test problems solved,
(3) there are few guidelines to reduce the computational burden or to maximize the benefits of empirical tests.

As outgoing chairman, I would like to make a recommendation for the future. First, an analogy with Architecture, Engineering and Cathedral Building in the Middle Ages. At that time the discipline of construction engineering had few principles beyond informed judgement. Occasionally, cathedrals collapsed as evidenced by the famous "accident" in Beauvis, France 1284. In contrast, today the engineering profession has replaced the guilds of Freemasons, and systematic analysis has subsumed much of what was judgemental. The important distinction is that a body of theory exists to guide the designer. Yet failures still occur--the Hartford coliseum.

As designers of computer algorithms, we are in a similar position to the Cathedral builders. Our computational results depend greatly upon informed judgements. And Freemasons abound who are willing to extall their success stories.

To improve this state of affairs, I propose that an organization entitled "Society for Testing Software" be established under the aegis of the National Bureau of Standards or similar "neutral" agency with support from MPS and related societies. This group will be responsible for coordinating computational analysis and, perhaps, carrying out empirical tests. It will fill a niche between academics who do not have enough time or monies to conduct deliberate empirical tests and code developers at software houses who are restricted because of propriety interests. This organization will be similar to the "American Society for Testing Materials" which performs a similar function for construction engineers.

Naturally, this organization will require a considerable budget and competent technical staff. And the benefits of an STS may not be greater than the costs until mathematical programming applications become even more pervasive. But such a day is fast approaching.

CALENDAR OF MATHEMATICAL PROGRAMMING MEETINGS AS OF DECEMBER 13, 1979
Maintained by the Mathematical Programming Society (MPS)

Note: This is the first appearance of the MPS Calendar. It is devoted to meetings specializing in mathematical programming or one of its subfields in the general area of optimization and applications, whether the Society is involved in the meeting or not. (These meetings are not necessarily "open".) Any one knowing of a forthcoming meeting not listed here is urged to inform the Chairman of the Executive Committee of the Society, Dr. A. C. Williams, Computer Science Department, Mobil Oil Co. Technical Center, Box 1025, Princeton, New Jersey 08540, USA; telephone 609-737-3000, extension 2342.

Substantial portions of regular meetings of other societies, notably ORSA, SIAM, and TIMS, are devoted to mathematical programming, and their schedules should be consulted.

1980

February 8: "Workshop on polynomial-time algorithms for linear programming," IBM Systems Research Institute, 205 E. 42 Street, New York, NY. Contact: Philip Wolfe, IBM Research Center 33-221, P. O. Box 218, Yorktown Heights, NY 10598, USA; telephone 914-945-1642. Organized by the Executive Committee of the Society.

March 3-7: Mathematical Programming Project of SHARE at Anaheim, California. Contact: Thomas White, Bel Air Research Center, Shell Development Company, P. O. Box 481 Houston, Texas 77029; telephone 713-663-2387.

March 3-7: "The 11th Southeastern Conference on Combinatorics, Graph Theory and Computing." Florida Atlantic University, Boca Raton, Florida. Contact: Prof. Frederick Hoffman, Dept. of Mathematics, Florida Atlantic University, Boca Raton, Florida 33431; telephone 305-395-5100 extension 2756, 2758.

May 1-2: "Second Symposium on Mathematical Programming with Data Perturbations," The George Washington University, Washington, D. C. Contact: Prof. Anthony F. Fiacco, Dept. of Operations Research, The George Washington University, Washington, D. C. 20052; telephone 202-676-7511.

May 12-16: "Third International Discussion Conference on Operational Research," Hove, England. Contact: Prof. Patrick Rivett, Mantell Building, University of Sussex, Brighton, Sussex BNI 9RF, England.

May 21-22: "Optimization Days" at Ecole des Hautes Etudes Commerciales, Montreal, Canada. (Abstract deadline January 31.) Contact: Prof. Alain Haurie, Service de lenseignment des Methodes Quantitatives, Ecole des Hautes Etudes Commerciale, 5255 avenue Decelles, Montreal, Quebec, Canada H3T 1V6; telephone 514-343-3801.

June 16-20: "Workshop in Numerical Methods for System Engineering Problems" in Lexington, Kentucky. Contact: Prof. Roger J. B. Wets, Department of Mathematics, University of Kentucky, Lexington, Kentucky 40506; telephone 606-257-2836. Sponsored by the MPS.

TABLE I
DISTRIBUTION OF RESPONSES TO SELECTED RESPONDENT PROFILE QUESTIONS

		NUMBER	PERCENT
1.	Current Employment (275 people gave 276 responses)		
	A. Academic	159	57.6
	B. Industrial	60	21.7
	C. Consulting	25	9.1
	D. Government	17	6.2
	E. Other Not-For-Profit	8	2.9
	F. Other	7	2.5
2.	Years of Experience with MP (278 people gave 278 responses)		
	A. 0-2	9	3.2
	B. 3-5	57	20.5
	C. 6-10	103	37.1
	D. 11-20	92	33.1
	E. 21+	17	6.1
3.	Usual Source of MP Software (277 people gave 587 responses)		
	A. Developed and Coded by Respondent	231	39.4
	B. Purchased from Computer Vendors	77	13.1
	C. Supplied Free by Computer Vendors	63	10.7
	D. Purchased from Code Developers Other Than Computer Vendors	78	13.3
	E. Supplied Free by Code Developers Other Than Computer Vendors	124	21.1
	F. Other	14	2.4
4.	Best Description of MP Interests (278 people gave 710 responses)		
	A. Algorithm Development	207	29.2
	B. Software Development	154	21.7
	C. Algorithm Evaluation	106	14.9
	D. Software Evaluation	82	11.5
	E. Software Procurement	30	4.2
	F. Software User	110	15.5
	G. Other	21	3.0
5.	Class of MP Problems For Which Performance Indicator Questions Are Answered (275 people gave 275 responses)		
	A. General Linear Programming	39	14.2
	B. Large Scale LP (>500 Constraints)	38	13.8
	C. Networks	25	9.1
	D. Transportation and Assignment	5	1.8
	E. Dynamic Programming	6	2.2
	F. Integer Programming	53	19.3
	G. General Nonlinear Programming	58	21.1
	H. Geometric Programming	6	2.2
	I. Unconstrained Optimization	21	7.6
	J. Other	24	8.7

somewhat different set of important indicators, including for example the design of the system (modularity, structured programming, etc.) and the input scheme.

Finally, Table IV presents the top five performance indicators with the respondents classified by their usual source of math programming software. The term "vendor" is used in the table to indicate "computer vendor" and "developer" refers to "code developers other than computer vendors." Again there is strong agreement among the categories, with documentation consistently ranked highest.

In summary, this survey served at least two purposes. It is clearly a first formal attempt to determine the preferences of the math programming community with respect to the important characteristics of the software being developed and utilized. It is worth emphasizing again that both CPU time and documentation are extremely important. The implications of this finding are clear: there is a growing need for accepted standard procedures which can be used to quantify these and other indicators of performance so that both developers and users can readily ascertain the merits of the software.

A second benefit of the survey is that it caused at least 269 people to spend some time thinking about the issues in this one aspect of math programming software evaluation. We appreciate the time and effort of those individuals in providing us with their opinions. Obviously, much work is needed to develop reasonable quantifiable measures of software performance. It is hoped that the results of this survey will serve to focus attention for future work in this area.

****

	ALL RESPONDENTS		GENERAL NONLINEAR		INTEGER PROGRAMMING		LINEAR PROGRAMMING		LARGE SCALE LP	
NO. OF RESPONSES	275		58		53		39		38	
	Rank	Score	Rank	Score	Rank	Score	Rank	Score	Rank	Score
CPU Time	1	1.16	1	1.26	1	1.62	4	.85	5	.79
Documentation	2	1.15	3	.98	4	.89	1	1.56	3	1.08
Rel. Dif. Obj. Fn.	3	.93	2	1.05	2	1.28	3	1.03	----	
Cost/Run	4	.89	----		3	1.07	2	1.10	1	1.32
Sensitivity Measures	5	.85	4	.90	----		5	.82	2	1.21
No. Funct. Evals.	----		5	.85	----		----		----	
Max. Rel. Error in Soln. Vec.	----		----		----		----		4	.90
Ease of Modification	----		----		5	.70	----		----	

TABLE II
TOP FIVE PERFORMANCE INDICATORS FOR ALL RESPONDENTS
AND FOR TOP FOUR MP CLASSES

July 7-11: "Tutorial Conference on Practical Optimization" in Stanford, California. Contact: Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, California 94305.

July 14-16: "Nonlinear Programming Symposium 4" in Madison, Wisconsin. Contact: Prof. Olvi Mangasarian, Computer Sciences Department, University of Wisconsin, 1210 W. Dayton Street, Madison, Wisconsin 53706; telephone 608-262-1204. Sponsored by the MPS.

July 22-25: "The Fourth European Congress on Operations Research" in Cambridge, England. COAL will sponsor at least one session on software testing at this meeting. For further information about the COAL sessions contact either Dr. Susan Powell or Dr. Jan Telgen (addresses on front cover of newsletter).
For general conference information, contact Prof. J. P. Brans, University of Brussels, VUB/CSOO. Pleinlaan 2, B-1050 Brussels, Belgium.

July 28-August 1: "Workshop on Mathematical Programming Software," SOGESTA, Urbino, Italy. Contact: Ditt. Claudio Sandi, IBM Italia-Centro Scientifico, Via Santa Maria, 67, 56100 Pisa, Italy.

September 15-17: "Second IFAC Workshop on Control Applications of Nonlinear Programming and Optimization." A short course on System Identification is scheduled immediately following the Workshop on September 17-19. (Abstract deadline February 20). Contact: Dr. Klaus Well, DFVLR, Oberpfaffenhofen, D-8031 Wessling, Federal Republic of Germany. Telephone (0 81 53) 28-435 or Dr. Herbert E. Rauch, Lockheed Research Laboratory, Dept. 52-56, Bldg. 201, 3251 Hanover Street, Palo Alto, California 94304, USA; telephone 415-493-4411 extension 45076

September 16-18: "6th International Seminar on Algorithms for Production Control and Scheduling," Karlovy Vary, Czechoslovakia. Contact: Ing. Jifi Kral, House of Technology, Gorkeho nam. 23, 11282 Praha 1, Czechoslovakia.

1981

January: A Workshop to be held in Boulder, Colorado (2 or 3 days). Organized by the Committee on Algorithms of the MPS. Contact: Mr. Richard H. F. Jackson, Center for Applied Mathematics, National Bureau of Standards, Washington, D. C. 20234; telephone 301-921-3855.

July 13-24: "NATO Advanced Research Institute on Nonlinear Optimization" Cambridge, England. Contact: Professor M. J. D. Powell, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Silver Street, Cambridge CB3 9EW, England. Sponsored by the MPS.

1982

Late summer: Eleventh International Symposium on Mathematical Programming in Bonn, Federal Republic of Germany. Contact: Professor Dr. Bernard Korte, Institut fuer Oekonometrie und Operations Research, Universitaet Bonn, Nassestrasse 2, D-5300 Bonn 1, Federal Republic of Germany. Official triennial meeting of the MPS.

THE KHACHIAN PHENOMENON

The excitement the press has stirred up about about the innocently titled paper, Reference 2 below, must be unique in the annals of applied mathematics. I am trying to collect everything that has been written about it, from research papers to press clippings, for the Mathematical Programming Society's archives and for possible discussion at the Workshop to be held on February 8 (see announcement, this bulletin). I would be most grateful for contributions to this collection: what I have so far is listed below. Please notify me of any new material or observations either by mail or orally.

One interesting bit of news for the specialist is that the algorithm Khachian uses is presented by N. Z. Shor, of the Institute of Cybernetics, Ukranian Acadademy of Sciences, Kiev, in Reference 1.

Philip Wolfe
IBM Research Center 33-221
P.O.B. 218, Yorktown Heights, NY 10598, U.S.A.
Telephone 914-945-1642, Telex 137456

Technical papers (chronologically, then alphabetically)

1. N. Z. Shor, "Cut-off Method with Space Extension in Convex Programming Problems". *Kibernetika* 13, Jan.-Feb. 1977, pp. 94-95 (translated as *Cybernetics* 13, 94-96).
2. L. G. Khachian, "A polynomial algorithm in linear programming". *Doklady* Akademiia Nauk USSR 244 (No. 5, February 1979) 1093-96 (translated as Soviet Mathematics *Doklady* 20, 191-194).
3. P. Gacs and L. Lovasz, "Khachian's Algorithm for Linear Programming". Distributed in preliminary form at the X International Symposium on Mathematical Programming in Montreal, Canada, 27-31 August 1979, and available as Report CS 750, Computer Science Department, Stanford University, Stanford, California 94305, U.S.A.
4. B. Aspvall and R. E. Stone, "Khachiyan's Linear Programming Algorithm". Report STAN-CS-79-776, November, 1979. Departments of Computer Science and Operations Research, Stanford University.
5. G. B. Dantzig, "Comments on Khachian's Algorithm for Linear Programming". Technical Report SOL 79-22, November 1979. Department of Operations Research, Stanford University.
6. W. Murray, "Ellipsoidal algorithms for linear programming". Working Paper 79-1, November 1979. Department of Operations Research, Stanford University.
7. M. W. Padberg and M. R. Rao, "The Russian method for linear inequalities". Graduate School of Business Administration, New York University, New York, NY 10006, U.S.A.
8. B. Randol, "A possible improvement of Khachian's algorithm in linear programming". Department of Mathematics, CUNY Graduate Center, November, 1979. 33 West 42 Street, New York, NY 10036, U.S.A.
9. J. B. Rosen and G. Frawley, "A computational test of Khachian's algorithm for linear inequalities". Technical Report 79-28, November 1979. Computer Science Department, University of Minnesota, Minneapolis, Minnesota 55455, U.S.A.

Less technical writings

10. G. B. Kolata, "Mathematicians amazed by Russian's discovery". *Science* 206 (2 November 1979), 545-46.
11. M. W. Browne, "A Soviet discovery rocks world of mathematics". New York Times, 7 November 1979, p. 1 et seq.
12. J. Friendly, "Shazam! A shortcut for computers". New York Times, 11 November 1979, p. E7.
13. C. R. Whitney, "Soviet mathematician is obscure no more". New York Times, 27 November 1979, p. C1.
14. M. W. Browne, "An approach to difficult problems". New York Times, 27 November 1979, p. C3.
15. Separate letters by H. P. Boas, G. B. Kolata, and G. Adomian under the heading "Linear programming discovery". *Science* 206 (30 November 1979), p. 1022.
16. J. Beeler, "Experts see DP promise in Soviet algorithm". *Computerworld,* 10 December 1979.
17. "Fascinatin' algorithm". *Scientific American,* "Science and Citizen" section, January, 1980.

PERFORMANCE MEASURES FOR EVALUATING MP SOFTWARE

Table I presents the distribution of responses to selected questions in the Respondent Profile section of the questionnaire. It is interesting to note that many (39%) of the respondents are using software that they themselves developed. Selected aggregations reveal that 24% of the respondents use software obtained from computer vendors and 26% use codes which are purchased from other software suppliers. Approximately 51% of the responses indicate an interest in the development of algorithms and software and it is reassuring to note that over 26% of the responses expressed an interest in algorithm and/or software evaluation. It is somewhat surprising that only 15% of the responses described their interests as software users.

Each respondent was requested to select one class of math programming problem to keep in mind while rating and ranking the performance indicators. The top four responses were (in order) general nonlinear programming, integer programming, general linear programming, and large-scale linear programming. The complete distribution of responses is given in section 5 of Table I.

The primary purpose of the survey was to determine the relative importance of various performance indicators for different classes of math programming software. After rating each indicator, the respondents were asked to select the five most important ones. Each of the 38 indicators was then given a score for various classifications of the respondents. The scores were obtained by the following procedure. Let R_c denote the number of respondents in category c. Let V_{ijc} be the number of votes by respondents in category c for indicator j as preference i, $i = 1,2,\ldots,5$, $j = 1,2,\ldots,38$. Then the score for indicator j from respondents in category c is S_{jc} which is calculated as

$$S_{jc} = 1/R_c \sum_{i=1}^{5} i\ V_{ijc},\ j = 1,2,\ldots,38 .$$

Table II presents the scores and rankings of all performance indicators which appeared as the top five choices from the entire population of respondents and categorized by the four most popular classes of math programming problems.

It is not surprising that CPU time was selected as the most important indicator by the overall population and that it appeared in the top five choices for <u>all</u> math programming categories except unconstrained optimization. The second <u>highest</u> rated indicator, documentation, was the only indicator which placed in the top five (actually top four) by all respondents under this classification scheme. As far as solution quality is concerned, there was more interest in the relative difference between the true and computed objective function values than in any measures of solution vector quality. However, except for responses associated with integer programming and general nonlinear programming, solution quality was generally ranked less important than documentation and the two popular measures of computational effort, CPU time and cost per run.

Table III presents the top five performance indicators when the respondents are categorized according to their current employment. Here it is seen that there is complete agreement among those in academic, industrial, and consulting environments regarding the selection of the top five indicators. It is interesting that documentation is rated most important by all respondents except those in industry and consulting who rate it fifth. It is also interesting that the 31 respondents in government or "other" environments identify a

RESULTS OF A SURVEY ON MP PERFORMANCE INDICATORS

Harlan P. Crowder, IBM Thomas J. Watson Research Center
Patsy B. Saunders, National Bureau of Standards

One important aspect of designing computational experiments to test math programming software is the determination of the factors upon which code performance will be evaluated. Although central processing unit (CPU) time is the most frequently utilized factor, there are many other performance indicators worthy of consideration. This article presents preliminary results of a survey designed to gather information from the math programming community regarding preferences among many proposed performance indicators.

The survey consisted of two sections, one which attempted to characterize the respondent and another which required evaluating and ranking performance indicators. The first section asked questions regarding the respondent's professional affiliation (academic, industrial, government, etc.), experience (years, previous involvement with computational tests, etc.), and specific areas of math programming interest (lp, networks, nonlinear, etc.).

In the second section of the survey the performance indicators were presented in four groups. The first group consisted of static characteristics of the software, i.e. aspects which can be examined without actually conducting a thorough computational experiment. The second group related to the amount of computational effort required in solving problems and the third group contained various measures of the quality of the solution, i.e. both the objective function value, and the solution vector. Finally, the fourth group identified certain auxiliary information for which respondents were asked to rate the importance. In each group the respondents were asked first to rate each indicator on a scale of one to five, one being very important and five being irrelevant. Then, at the conclusion of each group, the respondents were requested to select the most important factors in that group. The last question in the survey asked respondents to consider simultaneously all of the performance indicators in all four groups and select the five most important ones.

As novices in the art of survey design, we acknowledge the imperfections of our questionnaire. In particular, there were difficulties in analyzing the results of questions in which respondents were permitted to "write in" a performance indicator and then rate and possibly rank the write-in response. Because so many of these responses were unclear, inappropriate, or irrelevant, all such responses were grouped together under one category as "other." Despite this deficiency, however, we feel that the majority of the questions on the survey yielded valid results.

Approximately 1450 surveys were mailed in the spring of 1979 to the entire Mathematical Programming Society and the North American subset (U. S., Canada, and Mexico) of SIGMAP. Copies were also included in the June 1979 issue of the COAL Newsletter which is sent to all friends of COAL. A total of 278 surveys were returned by 269 respondents. (Several respondents submitted multiple surveys to indicate different performance measure preferences for different areas of math programming.)

MATHEMATICAL PROGRAMMING SOCIETY

OFFICE OF THE CHAIRMAN

Workshop on polynomial-time algorithms for linear programming

In response to the widespread interest in L. G. Khachian's work the Mathematical Programming Society is holding a Workshop on the topic on Friday, February 8, 1980, at IBM Systems Research Institute, 205 East 42 Street, New York City.

We invite all those who are presently working with the algorithm or on its development into a practical tool for optimization to come and share their ideas with one another. Knowledge of the basic algorithm, as presented by Gacs and Lovasz ("Khachian's Algorithm for Linear Programming", Report CS 750, Computer Science Department, Stanford University), or by N. Z. Shor ("Cut-off Method with Space Extension in Convex Programming Problems", Kibernetika 13, Jan.-Feb. 1977, pp. 94-95) will be assumed. While it is the Society's policy that its meetings be as open as possible, particularly to members of the Society, it is hoped that the merely interested will not attend, and that each participant will have some new experience, insight, or idea to offer.

Since quite a few people have been working on similar aspects of the procedure, the precise format of the workshop has not been fully determined. We expect to schedule some oral presentations, limited strictly to 10 minutes, and provide ample time for discussion. A transparency projector, the preferred means of display, will be available. No fee will be charged. Contact Michael Held at the Systems Research Institute, telephone 212-983-7245, if information about local arrangements is needed.

Please inform me by any method as soon as possible if you wish to come, and let me know what aspect of the topic you are working on. **Whether or not you plan to come, please send me as soon as possible any paper or information which might be of use to the attendees.** Shortly before the meeting I will prepare a schedule and an attendance list. Please mention this announcement to any interested parties who may not have seen it.

Philip Wolfe
IBM Research Center 33-221
P.O.B. 218, Yorktown Heights, NY 10598, U.S.A.
Telephone 914-945-1642, Telex 137456

THE MATHEMATICAL PROGRAMMING SOCIETY

ENROLLMENT

I hereby enroll as a member of the Society for the calendar year 1980.

PLEASE PRINT:

Name____

Mailing address____________________________________

____

____

____

My subscription to *Mathematical Programming* is for my personal use and not for the benefit of any library or other institution.

Signed____________________________

The dues for 1980 are:

42 Dollars (U.S.A.)
20 Pounds (U.K.)
68 Francs (Switzerland)
178 Francs (France)
76 Marks (Fed. Rep. Germany)
84 Guilders (Netherlands)

Please send this application with your dues to

The Mathematical Programming Society
c/o The International Statistical Institute
428 Prinses Beatrixlaan
2270 AZ Voorburg, Netherlands

CHAIRMAN'S COMMENTS

Since 1973, when the Committee on Algorithms was formed, its members have been active in many areas, including the development of guidelines for reporting results of computational experiments, the collection of test problems, the investigation of techniques for removing timing variability from computational results, the establishment of reliable performance evaluation criteria for comparing MP software, the development of a sound methodology for comparing MP software, and of course, the organization of this newsletter. Recent activities in some of these areas are reported in the current issue. Detailed reports were presented at the Tenth International Symposium on Mathematical Programming in Montreal last August, where we were encouraged by the Executive Council of the MP Society to continue our activities.

One of our immediate goals is to increase the international cooperation among the researchers in computational evaluation. This newsletter, of course, serves as an ideal mechanism for this international interchange of ideas. Another way is to participate in, and organize, international conferences on topics of interest to COAL. (A list of upcoming relevant conferences is given later in this issue.) We plan to continue this pattern of international cooperation in the pursuit of the COAL objectives which are listed on the last page of this issue. In fact, with the approval of the Executive Council of the MPS, we have reorganized COAL under one chairman, and discarded the previous structure with two branches. We believe this will facilitate the flow of information.

This issue of the newsletter is being sent to all members of MPS. If you would like to continue receiving it, please write to the editor, Dr. Karla L. Hoffman, to request that your name be added to the list of "Friends of COAL." Please note, however, that we encourage the Friends to contribute to the newsletter as much as possible. Thanks for continuing to participate.

Richard H. F. Jackson
Chairman of COAL

COMMITTEE ON ALGORITHMS of the MATHEMATICAL PROGRAMMING SOCIETY

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 serve as a forum for the Friends of COAL. Through an informal exchange of opinions, members have an opportunity to share their experiences. To date, our profession has not developed a clear understanding on 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.

Karla L. Hoffman
Center for Applied Mathematics
National Bureau of Standards
Washington, D. C. 20234

T. STEIHAUG
SCHOOL OF ORGANIZA. & MGMT
56 HILLHOUSE AVE.
NEW HAVEN, CT 06520

Mathematical Programming Society

Committee on Algorithms Newsletter

January 1980 Karla L. Hoffman, Editor

Contents: