A COMPUTATIONAL COMPARISON OF 2-PHASE ALGORITHMS AND A RECURSIVE QUADRATIC PROGRAMMING ALGORITHM

G. van der Hoek, Econometric Institute, Erasmus University Rotterdam

1. The design of computational experiments and the selection of test problems

Theoretically the performance of a general nonlinear programming algorithm can be described in terms of *convergence* and *rate of convergence*. Then for suitably chosen problems, the guaranteed convergence and/or bounds on the rate of convergence can be calculated.
However, it is not advisable to compare general algorithms solely on a theoretical basis, as the actual convergence behaviour is clearly problem dependent. In practical problems convergence is obtained frequently under violation of theoretically imposed conditions, such as convexity or differentiability. Besides that up to now no single nonlinear programming algorithm has proved to be superior to all other nonlinear programming algorithms for *every* testproblem and for *every* required accuracy. For instance algorithms based on penalty functions do not use explicitly the given linearity of constraints, hence we expect them to be inferior to projection-like methods in case of application to linearly constrained problems. On the other hand penalty function like methods are expected to behave better on problems which contain (high) nonlinearities. The comparison of the algorithms of this note is still less simple: the recursive quadratic programming algorithms apply linearisations of the first order conditions of penalty functions, while the 2-phase algorithms apply a quadratic loss function in phase I and a linear penalty-like term together with linearisations in phase II.
A theoretical comparison of algorithms should be supplemented by a *computational comparison*: the algorithms are applied to a set of representative test problems and the efficiency of various methods in solving these test problems is measured in terms of some performance indicators. The representativity of the set of test problems means that both theoretical and practical problems are considered; there should be a significant difference in degree of nonlinearity, dimension, number of constraints and number of active constraints at x*. To meet all these requirements, we selected a number of test problems from the standard literature such as Himmelblau (1972), Colville (1968), Bus (1976), Cornwell, Hutchison, Minkoff and Schulz (1978) and Hock and Schittkowski (1979). An advantage

of such a test battery is that a comparison with computational experiments of other researchers is possible at acceptable costs.
A drawback of generating test problems randomly is that the resulting 'random' problems can have special properties. This means that the randomly generated problems may not be as random as they are supposed to be. Recently Van Dam and Telgen (1979) reported on this phenomenon for the case of randomly generated polytopes.
The set of test problems selected can be found in appendix A of Van der Hoek (1980). It is divided into 2 classes: 11 linearly constrained nonlinear programming problems (class LC) and 13 nonlinearly constrained nonlinear programming problems (class NLC). The class LC represents the majority of programming problems evolving from business applications while the class NLC especially corresponds to programming problems in research, development and engineering. The results obtained illustrate that it is not trivial to predict the influence of the problem characteristics on the performance of algorithms: a few highly nonlinear constraints may cause more difficulties than many active (almost) linear constraints at x*.

2. Termination criteria

In order to compare the robustness and the efficiency of competetive algorithms for constrained nonlinear programming, we have to apply the same termination criteria to all those algorithms. A choice remains to be made from the following general criteria:

i (relative) objective function improvement
ii (relative) stepsize
iii acceptable constraint violation
iv satisfaction of the first- or second order Kuhn-Tucker conditions
v satisfaction of the Jacobian uniqueness conditions.

The criteria i and ii directly concern the iteratively generated sequence $\{x_k\}$ and the corresponding function values. Criterion iii is self-evident and especially concerns possible constraint violations in the iteration points of the recursive quadratic programming algorithms. For well-behaved problem functions criteria iv and v will be met to within a certain precision as soon as ii and iii are satisfied.
Hence we decided to apply ii and iii as termination criteria, which mean that x* is located with a prescribed relative accuracy and that the constraint violations, if any, do not exceed a predesigned bound.

This means for the implementation that we shall define an algorithm to be convergent if both the following criteria are satisfied:

ii $$||x_k - x_{k-1}|| \leq \varepsilon_1(||x_k|| + 1)$$

for some pregiven precision parameter $\varepsilon_1 > 0$

and

iii $$|c_i(x_k)| \leq \varepsilon_2(||x_k|| + 1)$$, for all currently violated constraints and for some precision parameter $\varepsilon_2 > 0$

The precision parameters used are $\varepsilon_1 = \varepsilon_2 = 10^{-5}$.
As overall convergence is a result of convergence of the algorithms that solve the reduced problems generated, we have to augment the criteria mentioned above by termination criteria for the line search, for the algorithm which solves a linearly constrained reduced problem etc.
The line search in the recursive quadratic programming approach is terminated as soon as the Goldstein and Price test is satisfied with $\sigma = 0.01$ or if the distance of the successively generated iteration points along the line is smaller than or equal to ε_4, with $\varepsilon_4 = 10^{-2}$.
This last criterion is also applied in the line search of the 2-phase algorithms, with $\varepsilon_4 = 10^{-4}$.
The linearly constrained algorithm which solves the reduced problems of the 2-phase algorithm, is terminated if the norm of the search direction p_k, which is a projection of $\nabla F(x_k)$, is small enough: $||p_k|| \leq \varepsilon_5$, with $\varepsilon_5 = 10^{-4}$ *and* at the same time the expected improvement if a constraint is dropped is less than ε_5.

3. Performance indicators

The computational experiments with the algorithms were performed on the DEC 2050 of the Erasmus University Rotterdam, using the FORTRAN-20, version 5 compiler under the operating system TOPS 20, version 3.
The computational comparison should be based on generally applicable, preferably machine independent performance indicators.
The general applicability means that indicators such as the number of iterations of the 2-phase algorithms or the number of line searches applied in a recursive quadratic programming algorithm are not suited for our purpose, as they are based on the special structure of a group of closely

IMPLEMENTING A MATHEMATICAL PROGRAMMING ALGORITHM: PERFORMANCE CONSIDERATIONS AND A CASE STUDY

Uwe Suhl
Freie Universitat Berlin
Fachbereich Wirtschaftswissenschaft
Fachrichtung Wirtschaftsinformatik
Garystrabe 21, D-1000 Berlin 33

After a general discussion of program performance and its influence factors we consider a case study. A specific version of TOYODA's heuristic algorithms for solving multidimensional knapsack problems was implemented in three different FORTRAN programs. Starting from a straightforward program, we introduced increasingly sophisticated data structures to improve the asymptotic computational time complexity of the programs. All programs were compiled with the enhanced version of the FORTRAN H compiler available on IBM/370, 43xx, 303x machines, using its four compiler options. A computational study was performed with a series of randomly generated test problems with up to 3200 variables and constraints and about 70000 nonzeros. The purpose of this study was to measure virtual central processor times and working set sizes as a function of problem size and to study the influence of code optimization performed by the compiler. Since all three programs are representations of the same mathematical algorithm, conclusions can be drawn as to the relative importance of the influence factors on program performance. As it is also demonstrated, the potential performance of a representation of an abstract algorithm may be difficult to project without a careful implementation.

COMPUTATIONAL METHODS FOR MINIMUM SPANNING TREE PROBLEMS

R. E. Haymond, J. P. Jarvis, Clemson University
D. R. Shier, National Bureau of Standards,
Center for Applied Mathematics
Washington, DC 20234

Minimum spanning trees, which connect together nodes of a network at minimum cost, find application in such diverse areas as distribution systems, clustering, pattern recognition, and reliability. This paper surveys the well-known procedures of Kruskal, Prim, and Sollin for constructing minimum spanning trees. Various data structures that can facilitate the implementation of these procedures are also examined in some detail.

A preliminary computational study has been performed using four variants of the Kruskal algorithm and two variants of the Prim algorithm. All six algorithms were coded in "structured" FORTRAN and were designed to be highly modular. Thus, all Kruskal-type algorithms were identical except for minor differences in the initialization section and more substantial differences in the "cycle-detection" section. A similar modular approach was used in designing the Prim-type algorithms.

The test problems used were randomly connected networks with 25-500 nodes and 25 to 14,000 edges. Network edge densities ranged from 10 percent to 100 percent, and edge costs were randomly generated from a uniform distribution over 1 to 10,000. Both CPU times (using the extended H compiler on an IBM 370/3033) and statement counts (using the WATFIV profiler) were collected during the test runs.

Computational results indicated that the Prim-type algorithms were superior to the Kruskal-type algorithms, even at low densities (a surprising result). The empirically-measured CPU time for the Kruskal algorithms was $O(m \log m)$, while the empirical behavior of the Prim algorithms was $O(n^2)$, where n is the number of nodes and m is the number of edges in the network. The fast algorithm (a Prim procedure) required no more than .21 seconds--which occured for a 300 node network with 13,455 edges. The performance of this prim algorithm, which is the only algorithm among those tested that is sensitive to the actual magnitudes of the edge costs (rather than simply their rank), was considerably enhanced in further tests on networks with a smaller range of edge costs.

A technical report that documents these findings can be obtained by writing to J. P. Jarvis, Department of Mathematical Sciences, Clemson University, Clemson, SC 29631.

related algorithms. Furthermore, the required machine independence is pursued to simplify comparison with experiments on other computers. A disturbing factor in the measurement of the indicators is the use of prior information which is not included in the statement of the problem, such as an initial value of the penalty parameter, the initialisation of the inverse Hessian approximation etc. We excluded these undesirable influences by using the same parameter values for *all* test problems, except 1 very unfavourable case. This means that all results concern one and the same implementation where no attempt is made to 'optimise' the parameter choices with respect to a particular problem. The obtained parameter choices are 'safe', in the sense that both the efficiency *and* the robustness of the resulting implementation are satisfactory with respect to the problems investigated.
Possible performance indicators are:

i "Ease of use" of the algorithm.
ii Number of failed runs or robustness of the algorithm.
iii The (standardised) number of CPU-secs, needed to solve problems.
iv The number of problem function evaluations needed to solve problems.

The first mentioned indicator *ease of use*, could be measured in terms of: preparation time for executing a problem, the difficulties met in diagnostic work to find the causes of failures, the possibility of human errors (e.g., in analytically supplied derivatives). Though these aspects should be reweighted in comparing algorithms, they are really programmer dependent as well, hence we consider them to be qualitative indicators which are hard to measure objectively.
The second indicator, *the number of failed runs*, is indicated by simply marking those runs by the character F in the corresponding position of the table of results.
The (standardised) number of CPU-secs can be regarded as an indicator of the 'total effort' to solve a problem. A part of this effort will be reflected in the number of problem function evaluations, the remaining part mainly concerns all kinds of calculations performed during the execution of the program. Usually the time required for I/O-generation is excluded from these figures.
The rationale for using *standardised* CPU-times was to eliminate machine dependent and environment dependent influences such as access to memory and multiprogramming facilities. Usually the standardisation of CPU-time

is performed by dividing by the number of CPU-secs required for the execution of Colville's standard timing program. Recent research shows that the desired machine independence of the resulting figures is not realised in this way, mainly because of factors such as the workload of the machine methods of timing and the use of optimising compilers (Eason (1977) and Hoffmann and Jackson (1979)). Moreover Himmelblau (1972) points out that Colville's program is not representative as a 'meaningful standard timing program would be one that somehow takes into account the polymorphic factors of the arithmetic logic, access to memory, storage capacity, allocation of central processing vs. peripheral processing times'. As a result we decided to apply as performance indicator the *number of problem function evaluations*. Obviously the execution of an algorithm requires the computation of the value of both the objective function $F(x)$ and the constraint functions $c_i(x)$ at intermediate points. Instead of presenting all counted problem function evaluations separately, we shall present the number of *equivalent function evaluations*, as suggested in Staha (1973). This means that all constraint function evaluations are to be converted into objective function evaluations. This conversion is realised using the estimated ratios of the costs of the constraint function evaluations and the concerning objective function evaluation at the point $x^T = (1, \ldots, 1)$. These estimated ratios evolve from the comparison of the required number of CPU-secs to perform 10^6 function evaluations.

4. Results and conclusions

The computational experiments concern the following algorithms:

I Recursive Quadratic Programming with the Oren-Spedicato switch II update formulae

II 2-Phase algorithm with complete linearisation

III 2-Phase algorithm with restricted linearisation.

The Recursive Quadratic Programming (RQP) algorithm minimises a quadratic approximation of the objective function subject to a local linearisation of the first order conditions of an exterior penalty function. The implementation uses a self scaling updating technique for the 2nd order information. The RQP-implementation applied numerical differentiation using forward difference quotients with step size $\varepsilon_i = 10^{-8}(|x_i| + 0.001)$ for $i = 1, \ldots, n$. The 2-phase implementations apply an exterior penalty step as phase I. The second phase consists of the minimisation of an appropriately defined auxiliary objective function subject to (a selection of) linearised

REFERENCES

[1] T. Cheung, "Computational Comparison of Eight Methods for the Maximum Network Flow Problem," Technical Report 78-07, Department of Computer Sciences, University of Ottawa, Ottawa, Canada, 1978.

[2] W. H. Cunningham, "A Network Simplex Method," Math. Programming 11 (1976) 105-116.

[3] L. R. Ford and D. R. Fulkerson, Flows in Networks (Princeton University Press, Princeton, N. J., 1962).

[4] F. Glover, D. Klingman, J. Mote, and D. Whitman, "Comprehensive Computer Evaluation and Enhancement of Maximum Flow Algorithms," Research Report CCS 356, Center for Cybernetic Studies, The University of Texas at Austin, and MSRS 79-1, College of Business, University of Colorado, Boulder, Colo., 1979.

Recent Computational Testing

Four of the most significant of the approximately 70 algorithmic implementations are treated in this note. The first, labeled FIFO, represents the "collective wisdom" of the numerous contributors to the classical label tree school of thought. This code scans the arcs incident to a node in a first labeled, first scanned order. The determination of the allowable flow augmentation is postponed until breakthough has occurred. In addition, after augmenting flow, a portion of the labels in the label tree are retained. The second code, referred to as MAXAUG, processes the nodes according to the largest augmentation criterion. That is, the node which allows the largest flow augmentation from the source node in the current label tree is processed first. An address calculation sort is used in order to efficiently carry out this ordering. Our implementation of this method makes it astonishingly more efficient than prior studies found it to be. The third code presented here, called SUBREF, is an implementation of what we called the subreferent algorithm, developed in this study, which turned out to be the best of the several referent-type algorithms tested. This algorithm departs from Dinic's method and its recent variants by keeping track of the referent (actually, only a subset) in an implicit manner that accelerates labeling and flow augmentation, while simultaneously avoiding blind alleys potentially encountered in other approaches. The final code, referred to as PRIMAL, is our best overall implementation of the specialized primal simplex algorithm. A special basis tree structure is used in order to enhance the solution capabilities of the code.

In terms of computer memory requirements, PRIMAL is the best of our 70 implementations. As indicated in Table 2, PRIMAL requires roughly one-third of the core storage of the other approaches. This suggests that the specialized primal simplex algorithm would be ideal for applications requiring the solution of a series of maximum flow problems within a larger master problem.

Relative solution efficiencies, discernible from the times tabulated in Table 1, show that the classical FIFO approach is strictly dominated often by all other approaches and always by at least two of the other three codes. Our implementation of the sub-referent algorithm is the best code for the problem topologies with a low degree of underlying structural specificity. A complete report on our algorithmic development and testing [4] can be obtained upon request from the authors.

Table 2

Required Number of Arrays*

Code	$\|N\|$	$\|A\|$	$\bar{c}$
FIFO	4	6	0
MAXAUG	5	6	1
SUBREF	5	6	0
PRIMAL	7	2	0

*$|N|$ node length array. $|A|$ arc length array. $\bar{c}$ maximum arc capacity length array.

Recent Computational Testing

constraints. The 2-Phase implementations behaved better with central difference quotients with $\varepsilon_i = 10^{-4}$. The results obtained are mentioned in the table in the columns indicated by I - III. The columns indicated by COMET, Lootsma, GREG and GPMNLC are taken from Staha (1973) who used Himmelblau's set of testproblems. Our problems 2, 7, 9, 12, 15, 16, 18, 19 and 20 are the same as problems 17, 10, 4, 24, 11-1, 7, 13, 9 and 18-1 respectively of Himmelblau's collection. Staha's figures concern experiments with numerical derivatives as well. The *COMET* algorithm was proposed in Staha (1973). It is a penalty function algorithm which applies moving truncations of the constraint set to control the convergence towards the optimum. *Lootsma's* implementation concerns a mixed interior point-exterior point penalty function algorithm which applies extrapolations to accelerate the convergence towards the optimum. *GREG* denotes the Generalised Reduced Gradient algorithm of Abadie and Guigou (1969). It is based on linearisations using constrained derivatives to project conjugate directions on the feasible set defined by the linearised constraints. Newton iterations are used as a restoration procedure. *GPMNLC* is an implementation of Rosen's Generalised Projection Method for Nonlinear Constraints. It applies Goldfarb's projection formulae for linear constraints with a quadratic loss penalty function for the nonlinear constraints.

Discussion of the computational results

The results summarised in the table are an indication of the robustness and the efficiency of the implementations of the algorithms.
The RQP-algorithm seems to have more difficulties to solve the test in a satisfactory way: two less attractive local solutions were reached (problems 11 and 21). However, it succeeded in determining in problem 17 a local solution that is overlooked by most algorithms. Probably this behaviour of RQP is due to the fact that it allows for infeasible iteration points. On the other hand, RQP is *more efficient*, both for the linearly constrained and the nonlinearly constrained problems. This result is rather surprising as far as the linearly constrained problems are concerned. It might be caused by the linearisation of the first order conditions of the exterior penalty function, which forms the basis of this algorithm. For the special case of testproblems with (almost) n active constraints at x^* (problems 3, 7, 8, 10, 11, 12, 14, 15, 16, 18, 20, 22 and 24) RQP clearly improves on the 2-phase algorithms.

The 2-phase algorithm II, which linearises *all* nonlinear constraints

at every major iteration of phase II, is *more robust* than RQP. This can be explained by the fact that it only uses feasible iteration points. The efficiency of this 2-phase algorithm is clearly improved if only the constraints of the current active set contribute to the definition of the linearly constrained reduced problem of phase II. This can be seen from the 2nd and 3rd column, especially for problems 13, 16, 18, 21 and 23. However, the use of such a simplified reduced problem gives rise to failures on problems 15 and 20. An explanation of this phenomenon is that now the reduced problem has a *fixed* active set. Hence it is not possible to use collected information on the status of the linearised constraints to adjust the active set of the reduced problem.

We recall from the conclusion of table 5.2 in Van der Hoek (1980) that *if* convergence is obtained, then algorithm III needs the *same* number of major iterations as algorithm II in which all nonlinear constraints are linearised. Algorithm III requires more major iterations *to detect* $I(x^*)$ but once it obtains $I(x_k) = I(x^*)$ its convergence is faster. This is in accordance with corollary 3 of theorem 4.7 of Van der Hoek (1980). This observation provides additional motivation to look for a phase I which end up with $I(z^*)$ as an active set.

When comparing our results with those in Staha (1973), we should be aware of some smal differences in the definition of equivalent objective function evaluations. The ratios which Staha uses to convert constraint function evaluations into objective function evaluations concern *groups* of constraints (e.g., *all* nonlinear constraints) whereas we calculated the ratios for *individual* constraints. Another disturbing effect is that Staha's ratios are based on the comparison of 1000 problem function evaluations. This leads to less accurate ratios (in our experiments with 1000 evaluations the ratios varied up to 10%). A last difference is that we did not count the evaluations of linear constraints.

As a result we have to be careful in drawing conclusions from a comparison with Staha's numbers. But it still seems to be justified to state that algorithms I, II and III, behave very well in comparison with COMET, Lootsma and GPMNLC. This conclusion seems not to be valid as far as GREG is concerned. Certainly for this case the relative results are disturbed by one more factor: algorithms I, II and III use *one* fixed set of parameter values to solve the *whole* test set whereas Staha reports that extended dignostic work was required to prevent failures for GREG. He reports that sometimes an artificial constraint, such as $\sum_{i=1}^{n} x_i \geq -1.0 \times 10^{10}$ had to be added or that the bounds in the problem formulation had to be narrowed.

The dense acyclic network problems possess the most elaborate structure. This class contains an even number of nodes. Every pair of nodes is connected by an arc directed from the node with the smaller node number to the node with the larger node number. The capacity of the arc (u,v) is 1 if $v > u + 1$ and is $1 + (u - |N|/2)^2$ if $v = u + 1$. Node 1 is the source node and node $|N|$ is the terminal node.

Although somewhat artificial, this class of problems was included because it was expected to require a large number of iterations (starting from a zero flow initial state) since the optimal solution is obtained when the flow on every arc in the network is at its upper bound.

To minimize the effects of sampling error, numerous test problems were generated for each problem topology and each set of problem dimensions. The times reported in Table 1 reflect the average solution time in c.p.u. seconds over all problems of the indicated dimensions.

Table 1. Average solution times*

Topology	NODES	ARCS	FIFO	MAXAUG	SUBREF	PRIMAL
Pure Random	250	1250	0.13	0.10	0.11	0.11
	250	1875	0.24	0.18	0.16	0.23
	250	2500	0.39	0.25	0.19	0.22
	500	2500	0.32	0.32	0.21	0.23
	500	3750	0.55	0.34	0.24	0.44
	500	5000	0.60	0.51	0.39	0.66
	750	3750	DNR	0.46	0.32	0.38
	750	5825	DNR	0.61	0.40	0.65
	750	7500	DNR	0.77	0.62	1.07
	1000	5000	DNR	0.43	0.38	0.63
	1000	7500	DNR	0.84	0.59	0.91
	1000	10000	DNR	1.01	0.68	1.72
Multi-terminal random	250	1250	0.55	0.39	0.18	0.29
	250	1875	2.78	1.14	0.37	0.63
	250	2500	2.51	1.74	0.36	0.62
	500	2500	0.97	0.72	0.29	0.35
	500	3750	2.80	1.42	0.46	0.94
	500	5000	8.49	6.03	0.69	1.63
	750	3750	DNR	0.85	0.48	0.75
	750	5825	DNR	2.28	0.65	1.17
	750	7500	DNR	6.50	1.00	2.75
	1000	5000	DNR	0.70	0.51	0.87
	1000	7500	DNR	1.99	0.78	2.19
	1000	10000	DNR	10.92	1.25	5.67
Multi-terminal grid	235	1240	1.07	DNR	0.32	0.26
	235	1640	1.29	DNR	0.27	0.20
	410	2120	3.37	DNR	0.60	0.54
	410	2720	3.13	DNR	0.49	0.49
	635	3200	DNR	DNR	0.92	0.86
	635	4000	DNR	DNR	0.85	0.73
	910	4480	DNR	DNR	1.16	1.14
	910	5480	DNR	DNR	1.30	1.29
Dense acyclic	20	190	0.20	DNR	0.17	0.06
	40	780	3.73	DNR	1.29	0.46
	60	1770	DNR	DNR	4.26	1.45
	80	3160	DNR	DNR	10.06	3.31
	100	4950	DNR	DNR	19.58	6.29

* In cpu seconds on The University of Texas' CDC 6600

RECENT COMPUTATIONAL TESTING

Extensive computational testing of the 70 algorithmic implementations (over all three classes of methods), was conducted on the University of Texas' CDC 6600 using the FORTRAN MNF Compiler. All codes were executed during periods of comparable demand for computer use and were implemented by the same systems analysts with no attempt to exploit machine hardware characteristics.

The primary objective of this research effort was to design a solution algorithm to solve the large-scale maximum flow problems that arise in the analysis of real-world transportation systems. In order to evaluate the solution capabilities of the numerous algorithmic variants and refinements that were studied, a data base of test problems was developed after consulting with a number of researchers in the network area on problem structures most appropriate for investigation.

Four distinct problem topologies were considered: (1) pure random, (2) multi-terminal random, (3) multi-terminal grid, and (4) dense acyclic. A uniform probability distribution was used in all instances to randomly select items such as nodes and upper bounds.

The random networks were constructed by initially identifying the node set N (whose elements may be assumed to be numbered from one to $|N|$). The arcs were generated by successively selecting ordered pairs of nodes, $u \in N$ and $v \in N - \{u\}$, thus creating an arc (u,v) for each pair selected. Multiple arcs directed from node u to node v were not allowed in this, or the other three, classes of test problems. The integer upper bounds, or arc capacities, were selected within a pre-defined interval. The source node, s, and the terminal node, t, are randomly chosen from the elements of N.

The multi-terminal random network class exhibits a small degree of underlying structure. The source and terminal nodes for these problems actually play the role of master source and master terminal nodes. This results by assigning infinite capacities to all arcs incident upon these two nodes, so that all nodes in the forward star of the source node serve as "effective sources," and all nodes in the reverse star of the terminal node serve as "effective terminals." The overall impact of this slight generalization of the random network structure is to create a problem that simulates a true multiple source and multiple terminal network.

In the transit grid class of networks, the source and terminal nodes again serve as a master source and a master terminal, implicitly creating a set of effective sources and effective terminals. All nodes other than s and t are referred to as grid nodes, which can be viewed as arranged in a rectangular grid of r rows and c columns. Every adjacent pair of grid nodes is connected by two oppositely directed arcs whose capacities are selected from a predefined interval.

Like the multi-terminal random class, this class of problems simulates multiple source and multiple terminal networks (and the algorithms are not amended to capitalize on this fact). The additional structure of the transit grid networks closely resembles that arising in urban transit planning networks. In this setting, the grid structure captures the form of transportation routes in the greater suburban area. Source nodes represent major transit exchanges or vehicle storage facilties and terminal nodes correspond to collection nodes which are connected to key demand points within the city.

RECENT COMPUTATIONAL TESTING

These results were obtained in close co-oporation with A.S. Louter and R.Th. Wymenga.

Table: Numbers of equivalent objective function evaluations

Algorithm	I	II	III	COMET	Lootsma	GREG	GPMNLC
Test problem							
1	53	62	62				
2	58	92	92	338	130	39	46
3	51	73	73				
4	319	269	269				
5	353	186	186				
6	38	42	42				
7	255	109	109	7235	2496	148	20
8	41	29	29				
9	525	727	727	9732	2861	350	134
10	159	350	350				
11	38d	43	43				
12	61	130	127	1326	959	314	599
13	282	1444	1371				
14	1150	1501	1598				
15	982	1463	F a	13482	7347	159	7098
16	11755g	7016	6958	3733	12614	699	8368
17	112e	541	545				
18	578	463	392	b	3865	285	12462
19	162	392	392	388	a	1656	c
20	5759	10630	F a	110775	46120	2876	12251
21	2621f	44475	43968				
22	4930	21609	21609				
23	210	1846	1660				
24	76	561	561				

Explanation of the abbreviations used

a: no solution reached

b : terminated at infeasible point

c : argument of exponent too large

d : local convergence to $(1, 1\frac{2}{3}, \frac{1}{3}, 0, \frac{1}{3}, 1\frac{2}{3})$

e : local convergence to (-.4536, 0.2105)

f : local convergence to (428.9, -31.1, -.47, 28.7, 149.5, 0.0, 11.5, 38.6, 0.0)

g : no Goldstein/Price test used because of discontinuities in problem functions.

Hence our general conclusion is that both the Recursive Quadratic Programming algorithm with Self Scaling Update's for the 2nd order information and the two 2-phase algorithms are robust and efficient algorithms with respect to the test set used.

REFERENCES

Abadie, J. and J. Guigou , 1969, Gradient réduit généralisé, Note HI 069/02, Electricité de France.

Bus, J.C.P., 1976, Private communication.

Colville, A.R., 1968, A comparative study on nonlinear programming codes. I.B.M. New York Scientific Centre, Tech. rep. 320-2949.

Cornwell, L.W., P.A. Hutchison, M. Minkoff and H.K. Schulz, 1978, Test-problems for constrained nonlinear mathematical programming algorithms, Argonne National Laboratory, Applied Mathematics Division, Technical Memorandum 320.

Eason, E.D., 1977, Validity of Colville's time standardization for comparing optimization codes, ASME Design Engineering Technical Conference, Chicago, September, 77-DET-116.

Himmelblau, D.M., 1972, Applied nonlinear programming, McGraw-Hill Book Company, New York.

Hock, W. and K. Schittkowski, 1979, Test examples for the solution of nonlinear programming problems, part I, II, preprint 44, Institut für Angewandte Mathematik und Statistik, Universität Würzburg.

Hoffman, K.L. and R.H.F. Jackson, 1979, Processing Time: An accurate measure of code performance? Paper presented at the Xth international symposium on mathematical programming, Montréal.

Stahl, R.L., 1973, Constrained optimization via moving exterior truncations, Ph.D. Thesis, University of Texas.

Van Dam W.B. and J. Telgen, 1979, Randomly generated polytopes for testing mathematical programming algorithms, Report 7929/O, Econometric Institute, Erasmus University Rotterdam.

Van der Hoek, J., 1980, Reduction methods in nonlinear programming, Mathematical Centre, Amsterdam.

AN EVALUATION OF MAXIMUM FLOW ALGORITHMS

Fred Glover, Professor of Management Science
University of Colorado, Boulder, Colorado 80309
Darwin Klingman, Professor of Operatons Research, BEB 608
University of Texas at Austin, Austin, Texas 78712
John Mote, Analysis, Research, and Computation, Inc.
P. O. Box 4067, Austin, Texas 78765

For a number of years the maximum flow network problem has attracted the attention of prominent researchers in network optimization. Since the groundbreaking work of Ford and Fulkerson [3], a variety of algorithms featuring good "worst-case" bounds have been proposed for this problem. Surprisingly, though, there have been almost no empirical evaluations of these algorithms.

Cheung [1] recently conducted the first significant computational investigation of maximum flow methods, testing several of the major approaches. Although an important step in the right direction, Cheung's implementations employ methodology and data structures originating at least a dozen years ago.

In the past decade, however, advances in network implementation technology have been dramatic. Sophisticated labeling techniques and more effective data structures have (a) decreased total solution time and/or (b) reduced computer memory requirements. As a result, widely held beliefs about which algorithms are best for particular problem classes have been steadily challenged and, in some cases, completely overturned. This study described in this note, likewise, was overdue. One of the major purposes of this study, therefore, has been to design and test maximum flow implementations that make the most effective use of the recent developments in network labeling and data organization techniques. To safeguard against being swayed too heavily by preliminary analyses (and past experience in other network settings), we implemented more than one type of data structure and associated processing technique for most of the algorithms tested.

During the course of our investigation we examined the two most widely heralded general classes of algorithms for maximum flow network problems--the label tree and referent algorithms. Over 50 codes were developed and at least partially tested for these methods. In the process we also developed and tested a new member of the referent class of algorithms, called the sub-referent method, which proved far more effective than all others.

In addition, we investigated a third type of approach which constitutes a special-purpose variant of the primal simplex method. Previously, researchers have neglected primal methods in favor of more classical labeling types of algorithms, first because the classical methods were obvious and "natural," and second because simple choice rules yield good worst-case bounds. Recently, Cunningham [2] has partly removed the theoretical bias against the class of primal simplex maximum flow methods by deriving a computational bound for one of its members (different from the method we developed). Although this theoretical bound is not nearly as good as those for other algorithms, practical experience in the network area over the past decade argues strongly for testing a derivative of the primal simplex methodology, since this type of approach has proved highly robust and effective in other network contexts. Over 20 implementations of our proposed variant of the primal method were tested utilizing alternative starts, pivot choice rules, and update techniques.

AN EVALUATION AND COMPARISON OF CURVE FITTING SOFTWARE

Rosemary E. Chang
Sandia Laboratories
Livermore, California 94550

The purpose of this study was to recommend software for approximating univariate functions from discrete data for the SLATEC Mathematical Subroutine Library, a cooperation among Sandia National Laboratories, Los Alamos National Scientific Laboratory, and the Air Force Weapons Laboratory. Capabilities to be provided were determined by the needs of the cooperating laboratories. The codes were tested in four categories, (1) interpolatory polynomials, (2) least squares polynomials, (3) interpolatory splines, and (4) least squares splines. Proprietary software from NAG, IMSL, and PORT were included in the testing solely for comparison. To facilitate the testing, a data set generator was developed to create test sets of specified shape, abscissae configuration, scaling, and noise. Over 7000 data sets were used on 27 subroutines. Evaluation criteria were formulated by balancing the need to provide both for the inexperienced and the sophisticated user. They include ease of use, error detection, performance, range of applicability, timing, and storage requirements. A series of charts summarize the testing and evaluation results. Six codes plus auxiliary routines were recommended on the basis of this evaluation and on the potential for remedying any deficiencies uncovered.

For more information on this research, contact Rosemary Chang at the above address.

COMPUTATIONAL EXPERIENCE WITH
ELLIPSOID ALGORITHMS FOR LINEAR PROGRAMMING

A. C. Williams
Mobil Technical Center, Box 1025, Princeton, N.J. 08540

1. INTRODUCTION

This paper reports on some numerical experiences with solving small linear programs using the ellipsoid methods which came into such prominance in 1979 when the popular press picked up on L. G. Khachiyan's remarkable achievement of showing that such methods can actually solve linear programs in polynomial time (2). The brief initial period of exagerated enthusiasm was quickly replaced by an exagerated pessimism, as initial computational tests proved very negative. As shown in this report, however, the performance of the ellipsoid method can, with a few modifications and a slight compromise in generality, be brought to a point that the comparison with the simplex method for some small practically oriented problems is down to a matter of a factor of 15 to 20 times, rather than the factors of thousands and millions that have been quoted. Moreover, for a class of problems specifically designed to frustrate the simplex method, the version of the ellipsoid method used here actually proved superior, indicating that, in principle at least, there could be specialized areas in which it might be useful for calculation.

It does not appear that an ellipsoid method will ever be a serious competitor to the simplex method as a general purpose routine for the solution of linear programming problems. The simplex method takes full advantage of linearity, and of the empirical fact that in problems of practical interest, the paths which follow from one basic solution to a next better one seem always to lead to an optimal solution in an efficient manner. The ellipsoid methods, on the other hand, ignore the linear structure, and would proceed as well, or almost, whether the constraints are linear or convex. In the computational experience reported upon here, the simplex method has proven vastly superior for the problems selected from the literature as somewhat representative of small practical problems. There is a production planning problem, a gasoline blending problem, and a transportation problem.

On the other hand, the efficiency of the simplex method does depend upon the basic solution paths being efficient. Klee and Minty (3) have shown that there are arbitrarily large problems for which the simplex method (without modification) would be doomed to search through every basic solution before arriving at an optimum. Chvatal then showed how to adapt the Klee-Minty result to generate, for each n, a linear program with n rows and n columns, which takes 2^n-1 simplex steps to solve. In this case the basic solution path of maximal local increase (selecting the largest dj), is not efficient, and for n greater than 10 or so, the ellipsoid methods presented here prove to be more efficient. The significance of even this somewhat modest victory is considerably mitigated by the fact that modifications of the simplex method that appear in all commercial codes overcome the difficulties posed by the Klee-Minty-Chvatal problems, and, in fact, solve them easily. Also, it is the case that in the range of problems studied here, the simplex method finds good approximations more efficiently.

2. THE ELLIPSOID METHOD

The notion of solving convex optimization problems by enclosing the feasible (and then the optimal) region within some large ellipsoid (or generalization thereof) and then shrinking it, appears to have been first introduced by N. Z. Shor in 1970. It was further developed by him and a number of other Soviet authors over the ensuing decade, culminating in Khachiyan's result. (See the bibliography by P. Wolfe(6)). The basic ellipsoid method for solving linear (or convex) constraint sets is this. Start with some point x, and construct an ellipsoid, P^0, with x^0 as center, large enough to contain some portion of the feasible set. At iteration k, check x^k for feasibility. If it is not feasible, select some constraint, r, which is violated and construct a hyperplane which separates x^k from the halfspace (or, in the general convex case, the convex set) determined by the constraint, r. The intersection of that hyperplane with the current ellipsoid is an ellipsoid, P', of dimension $n-1$. Determine the ellipsoid, P^{k+1}, which is tangent to P^k and which has P' as a subset. Then take x^{k+1} as the center of P^{k+1} and continue.

The salient points concerning the procedure are: (i) the set of feasible points contained in the initial ellipsoid, is contained in each subsequent ellipsoid, i.e., no feasible point is ever cut out, and (ii) the volume of the ellipsoid is reduced by at least a fixed factor depending only on n, the dimension of the space, at each iteration. This means that eventually x^k will be feasible, provided an interior solution of the feasible set exists.

For optimization, once a feasible point is found, the process can be continued as follows. If $f(x)$ is to be optimized, and the center, x, of the current ellipsoid is feasible, adjoin the constraint $f(x)>f(x)$, and continue. The sequence of feasible points so generated will converge to an optimal solution, since no optimal point is ever cut out, and the ellipsoids continue to shrink by a fixed fraction at each iteration.

This method appears to be similar to that given by Shor. It is only one method, however, and many variants are possible. Moreover, it is not yet an algorithm, since there are many possible ways to implement it.

In the Khachiyan version of the method for finding a feasible solution to linear inequalities, it is required that the input coefficients all be integers, and that all calculations be carried out with a precision of at least $T=23L+33nL$ bits (where L is the number of bits required to specify the input data, and T is, in general, truly large). The separating hyperplanes are taken as passing through the center of the current ellipse. The theoretical contribution is that, under these circumstances, and with a given starting ellipse, carefully spelled out by Khachiyan, either a feasible solution is obtained in $6n^2L$ iterations, or there is no solution.

It is important to realize that the particular ellipsoid algorithm described by Khachiyan, and later by Gacs and Lovasz (1), was a

Another possible solution to this problem of rapid dissemination of algorithms is, for example, to create another journal or perhaps a newsletter whose sole purpose would be to publish algorithms that are "experimental" in nature where there are no serious claims to accuracy, portability, or ease of use. I wonder if there are enough users and developers of algorithms out there who would be interested in seeing the creation of such a newsletter. I further wonder if we could ever find somebody who would be willing to take on the responsibility for organizing such a newsletter. And finally I wonder about whether, if we publish algorithms that are experimental, nonstandard, perhaps nonportable and perhaps therefore difficult to use, the community at large would find this a service.

Now I would like to move on to the other issue of subroutine libraries and collections whose primary purpose is to disseminate well-tested, well-documented usable codes that implement mathematical operations research algorithms. It seems to me that libraries have a responsibility to users as well as to developers of algorithms and mathematical software. What additional burdens would be placed on the creators and operators of such subroutine libraries if we were to allow experimental and perhaps incorrect or nonstandard algorithms to be included in their libraries and disseminated by them? I feel that very likely they would suffer if these codes and algorithms were distributed and potential users experience difficulties in getting them to work on the new systems. Where would the new users go to gather more information about this experimental code, how would questions about implementation be handled, and who would be responsible for debugging or correcting errors in these codes as they arise?

But perhaps I wander from the point a bit. As I understand Powell's argument, his experimental code was in fact included into both the Harwell and the Argonne libraries, but he felt that was not enough, and he wanted to see it distributed more widely. I can only recommend that, if including it in two of the well-known library collections of our profession was not enough and that Powell felt strongly that there are many other people in our profession that could benefit from it, and that therefore it was a valuable contribution to the literature of our society, it should then be cleaned up and added to the literature of our society, with all of the implications regarding rigor, accuracy, portability, and ease of use that implies.

To conclude, I would like to add that I feel one of the major problems of our profession these days is that we have for too long believed that producing codes that implement operations research algorithms is an endeavor that is somehow different from producing technical articles about our research regarding those algorithms. In my view, both of these efforts result in products of our profession, and both should be conducted with equal rigor and a view toward the historical record: both of these outputs should be subjected to intensive review before publication. I feel that only then will the computational side of our profession, which many agree these days is becoming more and more important, catch-up and bring to our profession the same reputation that our theoretical efforts have so far brought.

COMMENTS ON POWELL'S PAPER

Richard H. F. Jackson
National Bureau of Standards
Washington, D. C. 20234

I wish to take this opportunity to comment on the letter by Mike Powell entitled "Optimization Algorithms in 1979," which is published in this issue of the Newsletter. I feel the need to comment on that article because, while I am sympathetic to Powell's concerns as he has written them and especially with his desire to distribute his codes, I am afraid I do not agree with his conclusions and recommendations.

In his letter Powell mentions that there are two stumbling blocks to the rapid dissemination of his codes; one of them is the editorial policies and standards of the journals that publish algorithms, and the other is the standards of portability, ease of use, and documentation required by the various subroutine library collections. I would like to discuss each of these in turn, in light of Powell's recommendations.

In his letter Powell seems to feel that the standards required of algorithms before publication in a journal inhibit rapid dissemination. While this may be true, it seems to me that since a professional journal is an historical record of professional endeavor, we should always strive to maintain the highest standards for publication in those journals, and that publishing "experimental" codes that are incomplete or nonstandard or nonportable or nondocumented is at variance with this endeavor. I feel we should view these "algorithms" journals in the same light that we view our "technical article" journals, like Mathematical Programming or Operations Research. We should no more accept below-standard articles for one journal then we should for another. One way out of this, however, is that if one is unwilling to meet the standards for publication of an algorithm then one could distribute it oneself just as one would if he had produced a technical article that was never meant to be polished and published as part of an historical record.

But to conclude this point, I would like to stress that I do not think we should lower the standards for publication of the algorithms of our profession. I feel that this would be nothing more than a giant step backward. We have striven over the years to raise the level of the standards for publication of algorithms of our profession in order to solve the obvious problems that were created by the too-rapid dissemination of the too-incomplete algorithms that have been produced over the years. I would not want to see this changed and wiped away in the name of "rapid dissemination."

(As an aside, I would like to add that if one of the major impediments to publication of an algorithm is that the version of FORTRAN used in writing the algorithm or code is nonstandard and that the complaint is that it would take too much time to convert it to standard FORTRAN, there are computer-aided analysis techniques that can make this job much more easy. There are, for examples, verifiers that will take your code as input and give you a listing of each of the lines of nonstandard FORTRAN that exists, allowing the conversion process to go forth rapidly.)

theoretical device, and contained a number of features not relevant to actual computation. In particular:

1. Integer coefficients are not necessary. The ellipsoid method can be used for any rational input data.

2. The astronomical precision needed for the theoretical result is not needed. The variant of the ellipsoid method described below works well using rational approximations with usual orders of precision (in this case, the APL precision of about 16 significant digits).

3. If the inequality set is not consistent, it is not necessary to carry out the very large number of iterations given by the Khachiyan theory. If "deep cuts" are used infeasibilities are discovered fairly propitiously.

4. A factor contributing to the large number of iterations required by the Khachiyan method is that the estimate, as given by Khachiyan, of the size of the initial radius, is generally a vast overestimate. In most applications, bounds on the variables are easily obtained just from a knowledge of the problem, and in many cases, bounds can be obtained by simple manipulations of the constraints.

While the ellipsoid methods are not competitive with the simplex method, they are not as bad as some evaluations would have them, when these factors are taken into account.

3. ELLIPSOID ALGORITHMS.

To reduce these descriptions to an algorithm, we remark preliminarily that an ellipsoid in n-space is characterised by a point $\bar{x}$ (the center) and a positive definite matrix Q, in which case, the ellipsoid is the set of points x which satisfy

$$(x-\bar{x})\ Q^{-1}(x-\bar{x}) \leq 1.$$

An algorithm to find an interior solution to a set of linear inequalities is, based on the description by Gacs and Lovasz, as follows:

PROBLEM: To find a solution to $Ax<b$, where it is known that, if there is a solution at all, there is a solution x such that $0<x<X$. (A is mxn).

Initialize: Set $x^0=0$; set $Q^0=n\ I\ X$ (where I is the nxn unit matrix).

F: If $Ax < b$, stop; else choose the row r for which $A_r x^k - b_r$ is greatest.

Transform:

$$x' = x - (Qa)/(n+1)\sqrt{a^T Q a} \qquad \text{where} \quad a = A_r^T$$

$$Q' = (n/(n^2-1))(Q - 2(Qa)(Qa)^T/a^T Qa)$$

Go to F.

Recent Computational Testing

This is the algebraic formulation of the procedure described above.

Two modifications of this procedure are useful in actual computation. The modifications used here are due to P. Wolfe(7). First, calculation with Q leads to numerical instability very quickly. But, by factoring Q into J^TJ, and then applying the transformations directly to J, a numerically stable procedure is obtained. Secondly, the separating hyperplane which, in the Khachiyan method, passes through x^k, is a very bad choice. A better separating hyperplane to use is the violated constraint itself. This deep cut not only reduces the volume of the ellipsoid by the largest amount (for the given constraint), but it also makes possible the discovery of infeasibility in a timely manner.

The algorithm which incorporates these modifications becomes:

Initialize: Set $x^o=0$; set $J^o=\sqrt{n}I$ X.

F: If $Ax< b$, stop; else choose the row r s.t. $s=A_rx^k-b_r$ is greatest.

Transform:

$$a=A_r^T \qquad e=s/|Ja| \qquad \text{(if } e>1\text{, no solution exists)}$$

$$x'=x - (1+ne)\, J^TJa/(1+n)|Ja|$$

$$J'=n\sqrt{\frac{1-e^2}{n^2-1}}\left(I - \left\{1-\sqrt{\frac{(n-1)(1-e)}{(n+1)(1+e)}}\right\}\frac{(Ja)(Ja)^T}{Ja}\right) J$$

Go to F.

An algorithm to solve the linear programming problem is:

PROBLEM: To find an optimal solution to max cx, subject to $Ax\le b$, where it is known that, if there is an optimal solution at all, there is one such that $0\le x\le X$.

Phase 1: Find a solution, x^k, to $-cx<-V^o$, $Ax<b$, $-x<0$, (where V^o is given by $-\sum_K c_jx_j$, $K=\{j|c_j<0\}$), using the ellipsoid algorithm.

Phase 2: Reset V^k to $V^{k+1}=cx^k$, and find a feasible solution, x^{k+1} to the new constraint set, using the current (x^k,J) as the initial ellipsoid. The stopping rule is as follows.

The maximum of a linear function cx over the ellipsoid (x,J) is given by $cx-|Jc|$ and it occurs at $x+J^T(Jc)/|Jc|$. Therefore, if we start with an ellipsoid which contains an optimal solution (so that every succeding ellipsoid contains an optimal solution), an upper bound on the maximal value is available at each iteration. This provides an estimate of how close to optimal we are for each feasible point. One stopping rule could be to stop when the current value differs from the bound by a satisfactorily small amount.

Moreover, by calculating the maximal values of Ax and comparing with b, we can determine which of the constraints are still active. If the number of active constraints is reduced to n, an optimal basis has

Feature Article

[18] M.J.D. Powell, "A fast algorithm for nonlinearly constrained opt mization calculations", in Numerical Analysis, Dundee 1977, ed: G.A. Watson, Lecture Notes in Mathematics No.630, Springer-Verla (Berlin, 1978).

[19] M.J.D. Powell, "Quasi-Newton formulae for sparse second derivati matrices", Report No. DAMTP 79/NA 7 (University of Cambridge, 19

[20] M.J.D. Powell and Ph. L. Toint, "On the estimation of sparse Hes matrices", to be published in SIAM J. Numerical Analysis.

[21] K. Schittkowski, "Randomly generated nonlinear programming test problems and their application to compare optimization programs", Ninth IFIP Conference on Optimization Techniques (Warsaw, 1979).

[22] Ph.L. Toint, "On sparse and symmetric updating subject to a line equation", Mathematics of Computation, 31 (1977) 954-961.

****

COMMENTS ON POWELL'S PAPER

John M. Mulvey
Princeton University
School of Engineering/Applied Science
Princeton, NJ 08540

Besides providng an interesting survey of (un)constrained optimization methods, this paper raises a longstanding issue regarding the utility of standards. Certainly, standards, however they are enforced, have both benefits and costs. This point is especially pertinent when someone with Powell's stature decides that the current rules inhibit his research.

For various reasons, it appears that algorithm developers in certain areas of mathematical programming do not like to conduct comprehensive experiments. The results of the work might not be worth the toil, unfortunately.

Perhaps a compromise is possible. Computer programs could be designated as "prototypes" until the usual standards are met. The journals would continue publishing descriptions (mathematical) of the algorithms, indicating that a prototype program is available from the author, or perhaps from another source, such as Harwell.

Of course, the refinement of an idea may be more important than the inspiration. Many "algorithms" fail or succeed on the strength of the computer implementation. Take the case of linear network programs during the middle and late 1960's.

In fact, some important computer scientists believe that an algorithm remains undefined until, and unless, the computer implementation is presented in all its details.

References

[1] M.C. Biggs, "Constrained minimization using recursive equality quadratic programming" in Numerical methods for nonlinear optimization, ed: F.A. Lootsma, Academic Press (London, 1972).

[2] K. Brown, M. Minkoff, K. Hillstrom, L. Nazareth, J. Pool and B. Smith, "Progress in the development of a modularized package of algorithms for optimization problems" in Optimization in action, ed: L.C.W. Dixon, Academic Press (London, 1976).

[3] A. Buckley, "A combined conjugate-gradient quasi-Newton minimization algorithm", Mathematical Programming, 15 (1978) 200-210.

[4] R.M. Chamberlain, C. Lemaréchal, H.C. Pedersen and M.J.D. Powell, "The watchdog technique for forcing convergence in algorithms for constrained optimization", Tenth International Symposium on Mathematical Programming (Montreal, 1979).

[5] A.R. Colville, "A comparative study on nonlinear programming codes", Report No. 320-2949 (IBM New York Scientific Center, 1968).

[6] W.C. Davidon, "Optimally conditioned optimization algorithms without line searches", Mathematical Programming, 9 (1975) 1-30.

[7] W.C. Davidon, "Conic approximations and collinear scalings for optimizers", internal report (Haverford College, U.S.A., 1979).

[8] R.S. Dembo, "A set of geometric test problems and their solutions", Mathematical Programming, 10 (1976) 192-213.

[9] J.E. Dennis and J.J. Moré, "Quasi-Newton methods, motivation and theory", SIAM Review, 19 (1977) 46-89.

[10] R. Fletcher, "Minimizing general functions subject to linear constraints", in Numerical methods for nonlinear optimization, ed: F.A. Lootsma, Academic Press (London, 1972).

[11] R. Fletcher, "An ideal penalty function for constrained optimization", J. Inst. Maths Applics , 15 (1975) 319-342.

[12] D. Goldfarb, "Extension of Davidon's variable metric method to maximization under linear inequality and equality constraints", SIAM J. Appl. Maths, 17 (1969) 739-764.

[13] M.J. Hopper, "Harwell Subroutine Library: A catalogue of subroutines (1978)", Report No. R 9185 (AERE, Harwell, England, 1978).

[14] J.J. Moré, "Implementation and testing of optimization software", Report No. DAMTP 79/NA 4 (University of Cambridge, 1979).

[15] B.A. Murtagh and M.A. Saunders, "Large scale linearly constrained optimization", Mathematical Programming, 14 (1978) 41-72.

[16] M.J.D. Powell, "Quadratic termination properties of minimization algorithms I", J. Inst. Maths Applics , 10 (1972) 333-342.

[17] M.J.D. Powell, "Restart procedures for the conjugate gradient method", Mathematical Programming, 12 (1977) 241-254.

been determined, and exact primal and dual solutions can be obtained from a single inversion. This is the stopping rule used in the computer programs.

The above algorithm was coded in APL as SOLVE1.

A second algorithm, coded as SOLVE2, differs in two places from SOLVE1. First, the initial starting point is taken as $x = \frac{1}{2}X$, and $J = \frac{1}{2}\sqrt{n} \times I$. Second, in phase 2, instead of taking $v^{k+1}=cx$, we use a better estimate obtained as follows. Take v^{k+1} tentatively as the max of cx along the gradient direction from x^k. Also determine the max of cx along each of the coordinate directions from x^k, and choose v^{k+1} finally as the max of all these.

Our implementation of the ellipsoid method for the solution of the l.p. max cx, subject to $Ax \leq b$, $x \geq 0$, assumes that bounds are available for each of the variables, i.e., that $x \leq X$. In addition, it is assumed that: (i) The feasible set has a non-empty interior, and (ii) The optimal primal and dual solutions are unique.

Neither of the assumptions (i) or (ii) is essential for the method, but it was not considered worthwhile to expend the considerable extra effort that would be involved in producing a code that would allow for their elimination. Assumption(i) means that the numerical problems of staying on the boundry of the feasible set are avoided, while assumption (ii) means that, in fact, the number of active constraints will eventually be reduced to n without having to resort to some kind of perturbation scheme.

4. THE COMPUTATIONAL TESTS AND RESULTS

The tests were done on an IBM 3033U (MVS/APLSV), using the programs given in the appendix. In comparing the algorithms for the solution of the test problems, two measures were applied, namely, total iteration count, and monitor-reported CPU time. Each of these has its drawbacks.

The total iteration count has the advantage of being mainly independent of how carefully the coding was done, but the disadvantage of comparing items which are fundamentally not the same. A simplex iteration is only very roughly the same as an ellipsoid iteration, since the simplex method updates a matrix of size mxn, while the ellipsoid methods update matrices of size nxn. Moreover, in comparing different algorithms within the ellipsoid methods, we are concerned with a tradeoff between fewer iterations and more work per iteration.

The CPU time as measured by the APL $\square AI$ function also has its drawbacks. It can be as much a reflection of the programmer's skill as of the properties of the algorithm at hand. Moreover, the CPU time is not reproducible, since it depends on what other traffic was occuring within the computer at the time. We noted variations of up to 20%. Finally, the CPU time is not really the bottom line as a measure of performance, since there are other computer resources involved, notably memory. Tradeoffs between CPU time and memory are always involved in calculations of this type.

Recent Computational Testing

In the first series of tests, 3 problems were solved by the simplex method, and by the two versions of the ellipsoid method. The problems were:

1. AVGAS. A problem with 8 variables and 10 constraints (plus 8 non-negativity constraints). It is a simple model of the blending of aviation gasoline, taken from the book by G. Symonds, 1955 (5).

2. MPSX. A problem with 7 variables and 15 constraints (plus 7 non-negativity constraints, some of which are redundant). It is a simple model of the production of metal alloys, taken from IBM's MPSX manual.

3. TRANS. A transportation problem with 4 sources and 6 destinations. The resulting linear program has 17 columns (since not all source-destination pairs are feasible), and 10 constraints(plus 17 non-negativity constraints). It was taken from the MPSIII Manual, 1972.

4. TRANS2. The same as TRANS, except that the source 1 value is reduced so as to make the problem infeasible.

Complete specification of each l.p. used, along with the bounds, and the optimal solutions are given in the Appendix.

In the second series of tests, the Klee-Minty-Chvatal (KMC) problems of order 4 through 13 were solved by the simplex method, and by the two versions of the ellipsoid method. The KMC problem of order n is:

$$\max: \sum_{j=1}^{n} 10^{n-j}x_j; \text{ subject to } x_i + 2\sum_{j<i} 10^{i-j}x_j \le 10^{2i-2}, i=1,\ldots,n; \; x_j \ge 0.$$

Bounds are given by $x_j \le 10^{2j-2}$.

The appendix gives the results of applying the simplex method, and the two versions of the ellipsoid method to the first series of problems. While the simplex method required 9 iterations and .164 CPU seconds for AVGAS, SOLVE1 required 411 iterations and 4.957 CPU seconds, and SOLVE2 required 244 iterations and 2.754 seconds. The MPSX comparison is qualitatively the same, while for TRANS the ellipsoid method is even worse in comparison to the simplex method. For TRANS2, the infeasibilities are discovered in 258 and 210 iterations, respectively, as compared with 9 for the simplex method. One can conclude that the ellipsoid methods do not, and probably never will, compare favorably with the simplex method for this type problem.

In the second series of tests, the simplex method does better comparitively for small values of n, when results for obtaining exact solutions are compared. For values larger than 10 however the ellipsoid methods actually prove to be more efficient than the simplex, and for n=13, SOLVE1 obtains an exact solution in 3574 iterations and 58.5 CPU seconds, while the simplex method required 8191 iterations and 145.8 CPU seconds. While the iteration count and CPU time are increasing exponentially for the simplex method, they

Feature Article

Suppose first that the computer program is raised to an acceptable standard, not by myself, but by an expert in mathematical software. Then the delays that are mentioned in Section 2 are likely to occur, which would defeat the purpose of providing a computer listing in order that a *new* algorithm can be assessed properly. I am reluctant to try to meet the current standards of mathematical software, because I prefer my research to be on algorithms instead of on the details of computer implementations of algorithms. Further, in either case, the extra effort of meeting standards may be wasted, because the practical experience that is possible when a computer listing is available may show that extensive changes need to be made to the algorithm itself.

On the question of standards, it is important to retain the present quality of subroutines that are intended for general use on a wide range of computers. Therefore there are excellent reasons for introducing a new and more tolerant standard, in order that computer listings of new algorithms can be provided quickly and easily. I suggest that a new standard should require only that listings are free from errors in the language of the computer program, and that documentation is provided, written for people who have some experience of numerical calculations, that not only explains the use of the program, but also mentions any known deficiencies. Unless such a standard is accepted, I believe that pressure from the mathematical software community will tend to stifle the development of new algorithms, or will cause an increase in the proportion of algorithms that are proposed because of their theoretical interest.

Therefore the following aims seem to be sensible. The present standards should be applied to general library subroutines, and it should be accepted that these subroutines may not be available until five years after the underlying algorithm is published. A more tolerant standard should be accepted also, in order that authors of new algorithms can issue computer listings that allow their algorithms to be used and assessed. These aims would allow most researchers in optimization to contribute to their subject in a way that is of direct usefulness to the solution of practical optimization calculations.

disadvantages that have come from the use of the computer program are mentioned.

The advantages are strong. Many people have informed me that the program has solved constrained optimization problems, that are important to their work, more efficiently than other methods that they have tried. The computer listing has helped researchers to obtain practical experience of variable metric methods for constrained optimization. In particular, some of the interesting numerical results that are reported in Schittkowski [21] were calculated by a single length version of VF02AD. Even an unfavourable report that I received was advantageous, because it brought to my attention a limitation of the algorithm that I had overlooked. It came from Madsen and Pedersen of the Technical University of Denmark. They found a numerical example that shows that the final rate of convergence is sometimes only linear, due to the line search objective function that is used to force convergence from bad starting approximations. A suitable technique has now been developed to overcome the difficulty [4] and it will be included in subroutine VF02AD.

The main disadvantage of making available a computer program is that the deficiencies of the program are often assumed to be deficiencies of the algorithm itself. Therefore, before releasing VF02AD, I had to judge whether the known limitations of the subroutine, which have been mentioned already, would be so serious that the advantages of the algorithm would not be recognised. Fortunately it seems that I did make a good decision, except that many people have formed a view of the computing time of the algorithm, that may be unduly unfavourable, because most of the computer time of the present implementation is taken by the auxiliary subroutine for quadratic programming. I expect large gains in efficiency to be obtained by a more suitable quadratic programming method.

Because subroutine VF02AD has made a valuable contribution to research and to practical calculations, it is important to consider why I was unable to publish it in the algorithms section of a journal. The reason is that the subroutine does not meet the standards of portability, robustness and structure that are now expected in computer programs for general use. Thus we have a situation where a highly useful computer listing cannot be accepted by a journal because, if it is published, then adverse criticisms may be made of the journal because it does not maintain the present standards of mathematical software. Of course the way out of the dilemma is either to change the computer program or to change the standards. I will discuss both possibilities.

seem only to be increasing a little faster than linearly for the ellipsoid method. Note, however, that if approximations are satisfactory, the simplex method is still better. If, e.g., we are satisfied with a solution within to within 1 part in a million, the simplex method produces it, for n=13, in 512 iterations and 9.20 seconds of CPU time, while SOLVE2 requires 661 iterations and 11.43 seconds to find a solution to that accuracy. (The accuracy estimates using the simplex method were obtained using $cx+\Sigma\max(0,\delta_j)x_j$ as upper bounds on V at each iteration).

REFERENCES

1. P. Gacs and L. Lovasz, "Khachian's Algorithm for Linear Programming", Report CS 750, Computer Sciences Dept., Stanford University, 1979.

2. L. G. Khachiyan (Hacijan), "A Polynomial Algorithm in Linear Programming", Doklady 244(No. 5, Feb 1979) 1093-96. Translated as Soviet Mathematics Doklady 20, 191-194

3. V. Klee and G. J. Minty "How good is the Simplex Algorithm?" Inequalities III, ed. by O. Shisha, Academic Press, 1971.

4. N. Z. Shor, "Utilization of the Operation of Space Dilitation in the Minimization of Convex Functions", Kibernetica, 6 (No. 1, Jan-Feb 1970), 6-12. Translated as Cybernetics 6, 7-15.

5. G. H. Symonds, "Linear Programming: The Solution of Refinery Problems", Esso Standard Oil Co., New York, 1955, pp 18-24.

6. Philip Wolfe, "A bibliography for Ellipsoid algorithms", IBM Research Center Report, Feb 1979, revised Apr 1979.

7. Philip Wolfe, "The Ellipsoid Algorithm", Optima No. 1, June 1980.

A1

RECENT COMPUTATIONAL TESTING

```
      ∇SOLVE1[□]∇
    ∇ V←SOLVE1
[1]   ⍝ SOLVES THE L.P. MAX CX; AX≤B, X≥0; BY THE ELLIPSOID METHOD.(VERSION 1)
[2]   ⍝ INPUT GLOBALS: A B C BND II TITLE. (BND ARE THE UPPER BOUNDS ON X)
[3]   ⍝ OUTPUT GLOBALS: X S PI ΔJ K K1. SUBROUTINES USED: 0 NORM REPORT
[4]   ⍝ SET II←0 TO SUPPRESS LINE REPORT.  OTHERWISE SET II←1.
[5]   ⍝---SET INITIAL CONDITIONS AND CONSTANTS---
[6]    J←(N,N)⍴(BND×N*÷2),(N,N)⍴0 ⋄ X←N⍴0 ⋄ N←1↓⍴A ⋄ K←K1←K2←0 ⋄ T0←⎕AI[2]
[7]    VJ←+/(C>0)/BND×C ⋄ V←+/(C<0)/BND×C ⋄ AC←(N+⍴B)⍴1 ⋄ SOLVEN←'SOLVE1'
[8]   ⍝---SET UP INEQUALITIES---
[9]    AA←(-C),[1] A,[1]-(⍳N)∘.=⍳N ⋄ BB←(-V),B,N⍴0
[10]  ⍝ ---TEST FEASIBILITY---
[11]   E:→A×⍳0>MAXS ⋄ R←S⌷MAXS←⌈/S←(AA+.×X)-BB
[12]  ⍝---TRANSFORM X, TEST FOR NO SOLUTION, TRANSFORM J---
[13]   X←X-(÷N+1)×(1+N×E)×(⍉J)+.×JA←JS÷NORM JS ⋄ E←MAXS÷NORM JS←J+.×AA[R;]
[14]   ⍞(E≥1)/'→0 ⋄ V←''NO SOLUTION '',(▼K),'' ITS.'',''  TIME: '',▼⎕AI[2]-T0'
[15]   J←N×(((1-E*2)÷¯1+N*2)*÷2)×J-(1-((N-1)×(1-E)÷((N+1)×1+E))*÷2)×JA∘.×JA+.×J
[16]   →E ⋄ K←K+1
[17]  ⍝---NEW OBJ VALUE CALCULATION---
[18]   A:BB[1]←¯V←+/C×X
[19]   ⍞((K2=0)∨ΔV<10*-K2)/'K2←K2+1 ⋄ REPORT' ⋄ ΔV←(NORM J+.×C)÷|V+⎕CT×V=0
[20]  ⍝---DETERMINE NO. OF ACTIVE CONSTRAINTS. IF > N, GO TO E; ELSE GO TO E---
[21]   →E×⍳N<+/AC←AC×1↓0<S+NORM AA+.×⍉J ⋄ K1←K1+1
[22]  ⍝---CALCULATE PRIMAL AND DUAL SOLNS FROM BASIS---
[23]   E:V←+/C×X ⋄ S←,B-A+.×X←INV+.×BB[AC/1+⍳⍴AC] ⋄ INV←⌹AA[AC/1+⍳⍴AC;]
[24]   ⍞(II≠0)/'Z←REPORT' ⋄ PI←(-N)↓DUAL ⋄ ΔJ←(-N)↑DUAL÷AC\C+.×INV
      ∇
      ∇REPORT[□]∇
    ∇ Z←REPORT
[1]   ⍝ PRINTS LINE REPORT FOR "SOLVE ." USES SUBROUTINE STATS.
[2]    →(II=0)/0 ⋄ Z←0
[3]    ⍞(K2=0)/'⎕←STATS ⋄ ⎕←SOLVEN ⋄ ⎕←▼TITLE'
[4]    ⍞(K2=0)/'⎕←'' M IT    IT    AC     VALUE          ΔV      CPU TIME'''
[5]    ⍞(K2=0)/'⎕←(5 0▼(K1+1),K,+/AC),(¯10↑▼V),''      FEAS.'',¯10↑▼⎕AI[2]-T0'
[6]    →(K2=0)/0
[7]    (5 0 ▼(K1+1),K,+/AC),(¯10↑▼V),(10 ¯1 ▼10*-K2),¯10↑▼⎕AI[2]-T0
      ∇
      ∇STATS[□]∇
    ∇ Z←STATS;EE
[1]   ⍝REPORT STATS ON THE LP A B C
[2]    EE←+/,A≠0
[3]    Z←'ROWS:',(▼⍴B),' COLS:',(▼⍴C),' ELEM:',(▼EE),' DENS:',(▼EE÷×/⍴A)
      ∇
      ∇NORM[□]∇
    ∇ Y←NORM X
[1]   ⍝ NORM OF X (TAKEN OVER LAST COORDINATE)
[2]    Y←(+/X*2)*÷2
      ∇
      ∇O[□]∇
    ∇ C←A O B
[1]   ⍝ CATENATE SEPERATE STATEMENTS ON SAME LINE.
[2]    C←A
      ∇
```


FEATURE ARTICLE

4. The availability of computer subroutines

The numerical results of the last row of Table 1 were obtained by a computer program that I wrote in 1976. A slightly modified version of this program is now in the Harwell library, and it has the name VF02AD. It is instructive to consider the story of this subroutine, because of its relevance to the important question of providing computer listings of new algorithms for research and for the solution of real problems.

I coded the algorithm in Fortran IV because that was the most convenient language to use. The initial experimentation with the subroutine was exciting, because the number of function and gradient evaluations that were required to solve standard test problems was much fewer than had been reported for other methods. Experiments with bad starting approximations and highly nonlinear constraints were entirely successful, but I knew of the following limitations.

The computer program failed to solve one of Dembo's [8] problems. I believe that this failure was due to an auxiliary subroutine for quadratic programming, that was taken from the Harwell library to calculate the search direction of each iteration. Some simple pathological examples showed that this auxiliary subroutine sometimes loses accuracy unnecessarily, and also it has the disadvantage of always requiring lower and upper bounds on variables. Another limitation was that I had not given any attention to the possibility of saving computer time by making use of the relations between successive quadratic programming calculations. Moreover, I was sure that the algorithm would fail on some problems, because I had not included a suitable technique to force convergence to the solution of a system of nonlinear equations when the Jacobian matrix is singular, which may be needed when the number of active constraints is equal to the number of variables. I had not investigated the behaviour of the algorithm when the required solution $\underline{x}^*$ is on the boundary of some redundant constraints. Finally, I knew that some parts of the computer listing were in conflict with the rules of standard Fortran.

In spite of these limitations, I decided that the numerical results were so good that I would make the computer program available generally. Therefore I offered it to the Harwell library, to the Argonne National Laboratory, and for publication in the algorithms section of a well known journal. Both Harwell and Argonne accepted the program gratefully, but the journal refused to consider it because of the departures from standa Fortran. Before discussing the importance of standards, the advantages

after the linear approximations to the active constraints are satisfied. Therefore one can take the point of view that the variable metric algorithm solves one unconstrained problem in only four variables, while the augmented Lagrangian method has to solve a sequence of unconstrained calculations, where each one has fifteen variables. This view of the variable metric algorithm, however, is an over-simplification, partly because the algorithm does not identify the active constraints correctly until the eleventh iteration. The table below gives the number of times all functions and their first derivative vectors are calculated by three algorithms, in order to solve each of the three problems to five decimals accuracy. We note that the variable metric algorithms are much more efficient than the augmented Lagrangian method. Schittkowski [21] found similar gains in efficiency on a wide class of test problems, that was chosen to test the effects of zero Lagrange multipliers and indefinite second derivative matrices.

Table 1.

The number of points at which functions and first derivatives were calculated to solve three test problems

Method	Colville 3	Colville 1	Colville 2
Aug. Lagr. [11]	64	39	149
V. Metric [1]	10	8	47
V. Metric [18]	3	6	17

A2

Recent Computational Testing

```
      ∇SOLVE2[⎕]∇
     ∇ V←SOLVE2
[1]   ⍝ SOLVES THE L.P. MAX CX; AX≤B, X≥0; BY THE ELLIPSOID METHOD.(VERSION 2)
[2]   ⍝ CONDITIONS,SYNTAX,INPUT-OUTPUT SAME AS ∇SOLVE1.
[3]   ⍝---SET INITIAL CONDITIONS AND CONSTANTS---
[4]    J←(N,N)⍴0.5×(BND×N*÷2),(N,N)⍴0 ◇ X←BND÷2 ◇ N←1↓⍴A ◇ K←K1←K2←0 ◇ T0←⎕AI[2
[5]    VU←+/(C>0)/BND×C ◇ V←+/(C<0)/BND×C ◇ AC←(N+⍴B)⍴1 ◇ SOLVEN←'SOLVE2'
[6]   ⍝---SET UP INEQUALITIES---
[7]    AA←(-C),[1] A,[1]-(⍳N)∘.=⍳N ◇ BB←(-V),B,N⍴0
[8]   ⍝ ---INITIALIZE FOR "NEW OBJ VALUE CALCULATION" (STEP A BELOW)---
[9]    M←(M0←M>0)×M←(AA+.×C),((C>0)/AA),-(C<0)/AA ◇ C1←(+/C×C),((C>0)/C),-(C<0)
[10]  ⍝ ---TEST FEASIBILITY---
[11]   E:→A×⍳0>MAXS ◇ R←S⍳MAXS←⌈/S←(AA+.×X)-BB
[12]  ⍝---TRANSFORM X, TEST FOR NO SOLUTION, TRANSFORM J---
[13]   X←X-(÷N+1)×(1+N×E)×(⊗J)+.×JA←JS÷NORM JS ◇ E←MAXS÷NORM JS+J+.×AA[R;]
[14]   ⍎(E≥1)/'→0 ◇ V←''NO SOLUTION '',(⍕K),'' ITS.'','' TIME: '',⍕⎕AI[2]-T0'
[15]   J←N×(((1-E*2)÷¯1+N*2)*÷2)×J-(1-((N-1)×(1-E)÷((N+1)×1+E))*÷2)×JA∘.×JA+.×J
[16]   →E ◇ K←K+1
[17]  ⍝---NEW OBJ VALUE CALCULATION---
[18]   A:BB[1]←-V←(+/C×X)+⌈/C1×⌊≠(([/,W)×~M0)+W←(M0×⊗(⌽⍴M)⍴-S)÷M
[19]   ⍎((K2=0)∨ΔV<10*-K2)/'K2←K2+1 ◇ REPORT' ◇ ΔV←(NORM J+.×C)÷|V+⎕CT×V=0
[20]  ⍝---DETERMINE NO. OF ACTIVE CONSTRAINTS. IF > N, GO TO E; ELSE GO TO E---
[21]   →E×⍳N<+/AC←AC×1↓0<S←NORM A1+.×⊗J ◇ K1←K1+1
[22]  ⍝---CALCULATE PRIMAL AND DUAL SOLNS FROM BASIS---
[23]   E:V←+/C×X ◇ S←,B-A+.×X←INV+.×BB[AC/1+⍳⍴AC] ◇ INV←⌹AA[AC/1+⍳⍴AC;]
[24]   ⍎(II≠0)/'Z←REPORT' ◇ PI←(-V)↓DUAL ◇ ΔJ←(-N)↑DUAL←AC\C+.×INV
      ∇
EXAMPLE :
      A← 3 2 ⍴ ¯1 ¯1 1 2 1 ¯1
      B← ¯1 4 3
      C← 3 5
      BND← 4 2
      TITLE←'LP TEST'
      II←0
      SOLVE2
11.67
      8 3 ▼ 2 5 ⍴X,S,ΔJ,PI
   3.333    .333   2.667    .000    .000
    .000    .000    .000   2.667    .333
      II←1
      SOLVE2
LP TEST
SOLVE2
ROWS:3 COLS:2 ELEM:6 DENS:1
 M IT   IT   AC    VALUE       ΔV     CPU TIME
    1    1    5    11.33     FEAS.         24
    2    7    5    11.65     1E¯01         70
    3   16    3    11.67     1E¯02        134
    4   16    2    11.67     1E¯03        144
11.67
```

Recent Computational Testing

A3

```
AVGAS                    ROWS:10 COLS:8 ELEM:30 DENS:0.375

   0    2    1    3    2    4    3    5 = MAX

   1    1                                   ≤    1
             1    1                         ≤    1
                       1    1               ≤    1
                                 1    1 ≤    1
   1         1         1         1         ≤    2
        1         1         1         1 ≤    2
  -2        -1                   1         ≤    0
  -5        -3         3         1         ≤    0
            -1         1         3       5 ≤    0
            -1                   3       2 ≤    0

   1    1    1    1    1    1    1    1 :BOUNDS

VALUE: 7.75
SOLUTION
     1.00  .75  .25  .50  .25  .75
```

```
MPSX                     ROWS:15 COLS:7 ELEM:55 DENS:0.5238

  -.03  -.08  -.17  -.12  -.15  -.21  -.38 = MAX

 1.00  1.00  1.00  1.00  1.00  1.00  1.00 ≤   2000
  .15   .04   .02   .04   .02   .01   .03 ≤     60
  .03   .05   .08   .02   .06   .01       ≤    100
  .02   .04   .01   .02   .02             ≤     40
 -.02  -.03                 -.01          ≤     30
 -.70  -.75  -.80  -.75  -.80  -.97       ≤  -1500
 -.02  -.06  -.08  -.12  -.02  -.01  -.97 ≤   -300
 -.02  -.06  -.08  -.12  -.02  -.01  -.97 ≤   -250
            -1.00                        ≤   -400
                  -1.00                  ≤   -100
 1.00                                    ≤    200
       1.00                              ≤    750
             1.00                        ≤    800
                   1.00                  ≤    700
                         1.00            ≤   1500

  200   750   800   700  1500  2000  2000 :BOUNDS

VALUE: -296.2
SOLUTION
    .0 665.3 490.3 424.2    .0 299.6 120.6
```


$$c_i(\underline{x}) + \underline{\delta}^T \underline{\nabla} c_i(\underline{x}) = 0, \quad i = 1,2,\ldots,m' , \tag{3.6}$$

and

$$c_i(\underline{x}) + \underline{\delta}^T \underline{\nabla} c_i(\underline{x}) \geq 0, \quad i = m'+1,\ldots,m . \tag{3.7}$$

Therefore the calculation of $\underline{\delta}$ is a quadratic programming problem. This remark is an essential part of an algorithm (see [18], for instance), tha estimates B automatically, and that uses a line search procedure to force convergence when the initial estimate of $\underline{x}^*$ is poor. It is even possible to keep B positive definite, and then it is appropriate to call the algorithm a variable metric method for constrained optimization. Biggs [1] , however, prefers the name REQP (recursive equality quadratic programming).

Although the solution of a quadratic programming problem on each iteratio of a variable metric method for constrained optimization requires much more work than the calculation of the search direction $\underline{\delta}$ when no constraints are present, a variable metric method is often substantially faster than an augmented Lagrangian algorithm (see [21], for instance). The reasons are due to the fact that variable metric methods take account of linear approximations to the constraints. These approximations avoid the need for good estimates of Lagrange multipliers and for any penalty parameters, except perhaps in a line search objective function. Therefore it is no longer necessary to solve a *sequence* of general optimization problems. Another advantage is that conditions (3.6) and (3.7) cause the vector $\underline{\delta}$ to be less dependent on the matrix B that is estimated automatically. Three examples from Colville [5] make this important point clear.

In the "Colville 3" problem there are five variables and sixteen constraints, five of which are active at the solution. Hence, when the activ constraints are identified, which happens immediately because of the closeness of the standard starting point to the solution, the conditions (3.6) and (3.7) determine $\underline{\delta}$ uniquely. Therefore, in constrast to the augmented Lagrangian method, any errors in B make no difference. The variable metric method reduces to Newton's method for satisfying the active constraints and two iterations are sufficient to obtain five decimal accuracy in the calculated value of $\underline{x}^*$. Because the "Colville 1" proble has five variables and four active constraints, the matrix B controls only one degree of freedom in the search direction. In the "Colville 2" problem there are fifteen variables, but only four degrees of freedom

that usually the size of r remains moderate, there is a good version of the method that takes account of inequality constraints, and, even when derivatives are not calculated, there are suitable methods for adjusting λ. Further information can be found in the paper by Fletcher [11].

The algorithms that have been mentioned so far suggest that minimizing a function subject to nonlinear constraints is much more difficult than an unconstrained calculation in the same number of variables. This observation seemed until recently to be inevitable, because of the crucial effect that constraint curvature can have on the solution of an optimization problem. A two dimensional example that makes this point rather well is the minimization of x_2 subject to the condition

$$x_1^2 + x_2^2 = 1 \ . \tag{3.4}$$

It shows that the solution is unique only because the constraint is nonlinear. Because it is also easy to find examples where it is necessary to take account of the curvature of several constraints, it seemed ten years ago that the techniques of linear and quadratic programming were of little help to the general case. A simple remark, however, is helping to consolidate the techniques of mathematical programming in a way that treats nonlinear constraints very successfully.

The remark is that, if the second derivative matrix of the Lagrangian function (3.3) is known, and if first derivatives can be calculated, then, except in degenerate cases, it is possible to adjust an estimate of the required solution $\underline{x}^*$ in a way that has second order convergence. For example, Newton's method can be applied to the system of equations that is obtained from expression (3.1), and from the condition that the gradient of the Lagrangian function is zero at $\underline{x}^*$. The remark is important, because it shows that only one matrix of second derivative information is needed, even when several constraints are nonlinear. If B is the second derivative matrix of the Lagrangian function, and if $\underline{x}$ is an approximation to $\underline{x}^*$, then a convenient form of Newton's method for solving the problem given at the beginning of Section 1 is to replace $\underline{x}$ by $(\underline{x} + \underline{\delta})$, where $\underline{\delta}$ minimizes the function

$$\underline{\delta}^T \nabla F(\underline{x}) + \tfrac{1}{2}\underline{\delta}^T B \underline{\delta} \ , \tag{3.5}$$

subject to the constraints

A4

Recent Computational Testing

MPSIII TRANS PROB ROWS:10 *COLS*:17 *ELEM*:34 *DENS*:0.2

-8.0	-4.5	-4.6	-6.3	-6.0	-6.5	-8.5	-5.0	-7.0	-7.5	-7.6	-3.5	-7.9	-7.2	-4.8	-6.0	-5.0	= *MAX*
1				1			1		1						1		≤ 35
	1				1			1		1		1					≤ 35
		1				1					1		1				≤ 25
-1	-1	-1	1 -1											1		1	≤ 15 ≤ -32
				-1	-1	-1											≤ -18
							-1	-1									≤ -5
									-1	-1	-1						≤ -12
												-1	-1	-1			≤ -20
															-1	-1	≤ -13
32	32	25	15	18	18	18	5	5	12	12	12	20	20	15	13	13	:*BOUNDS*

VALUE: -505.5
SOLUTION:

	32				17	1		5				12			5	15	13

TRANS2: SAME AS MPSIII TRANS, EXCEPT THAT B[1] IS 24 INSTEAD OF 35. IT HAS NO FEASIBLE SOLUTION.

KMC ORDER 6 *ROWS*:6 *COLS*:6 *ELEM*:21 *DENS*:0.5833

1	10	100	1000	10000	100000	=	*MAX*
1	20	200	2000	20000	200000	≤	1E10
	1	20	200	2000	20000	≤	1E08
		1	20	200	2000	≤	1E06
			1	20	200	≤	1E04
				1	20	≤	1E02
					1	≤	1E00
1E10	1E08	1E06	1E04	1E02	1E00	:*BOUNDS*	

VALUE: 1E10
SOLUTION:
1E10

A5

Recent Computational Testing

PROBLEM STATISTICS

	AVGAS	MPSX	TRANS	TRANS2
ROWS	10	15	10	10
COLS	8	7	17	17
ELEMENTS	30	55	34	34
DENSITY	.375	.524	.200	.200

RUN RESULTS

ITERATIONS

SIMPLEX (PHASE I)	0	3	7	9
SOLVE1 (PHASE I)	24	30	104	258
SOLVE2 (PHASE I)	9	19	34	210
SIMPLEX (OPT)	9	10	8	
SOLVE1 (OPT)	411	534	3279	
SOLVE2 (OPT)	244	410	2575	

CPU TIME (IBM 3033 SEC.)

SIMPLEX (PHASE I)	.030	.110	.140	.163
SOLVE1 (PHASE I)	.247	.280	2.270	5.050
SOLVE2 (PHASE I)	.127	.200	.773	4.340
SIMPLEX (OPT)	.164	.233	.153	
SOLVE1 (OPT)	4.957	5.766	86.240	
SOLVE2 (OPT)	2.754	4.253	57.223	

KLEE-MINTY-CHVATAL PROBLEM OF ORDER:

		4	5	6	7	8	9	10	11	12	13
ITERATIONS											
SIMPLEX	$1E^{-3}$			32	32	32	32	64	64	64	64
SOLVE1	$1E^{-3}$	86	119	140	147	159	165	184	199	230	255
SOLVE2	$1E^{-3}$	67	77	98	113	131	122	128	122	154	212
SIMPLEX	$1E^{-6}$						256	512	512	512	512
SOLVE1	$1E^{-6}$			390	584	654	708	747	819	880	894
SOLVE2	$1E^{-6}$					474	515	545	566	650	661
SIMPLEX	EXACT	15	31	63	127	255	511	1023	2047	4095	8191
SOLVE1	EXACT	95	217	390	619	932	1342	1883	2526	3307	4248
SOLVE2	EXACT	67	123	258	446	718	1037	1531	2058	2801	3574
CPU TIME (IBM 3033 SEC.)											
SIMPLEX	$1E^{-3}$			.374	.407	.430	.460	.947	.970	1.040	1.196
SOLVE1	$1E^{-3}$	.683	.980	1.273	1.490	1.165	1.994	2.357	2.750	3.910	4.850
SOLVE2	$1E^{-3}$	.503	.633	.874	1.070	6.703	1.394	1.700	1.717	2.477	3.500
SIMPLEX	$1E^{-6}$						3.447	7.357	7.48	8.03	9.20
SOLVE1	$1E^{-6}$					5.806	1.384	8.760	9.473	11.46	15.51	17.71
SOLVE2	$1E^{-6}$					4.180	4.937	5.804	7.093	7.78	10.39	11.43
SIMPLEX	EXACT	.160	.333	.700	1.513	3.200	6.84	14.62	29.77	63.35	145.8
SOLVE1	EXACT	.767	1.770	3.593	5.186	9.526	16.56	28.50	34.73	56.33	81.2
SOLVE2	EXACT	.513	1.003	2.277	4.210	7.437	11.66	19.68	28.03	42.93	58.5

Feature Article

reduced gradient method. It is used often, but it is sometimes rather inefficient, due partly to the need for an iterative procedure to satisfy any nonlinear constraints in the active set. Because of the rapid development of new algorithms, I expect the present computer programs for the reduced gradient method to become obsolete in the next ten years, but I am sure there is a healthy future for active set strategies that can accept first order approximations to nonlinear constraints.

In my opinion the penalty function methods that do not make use of Lagrange parameter estimates should already have come to the end of their useful life. This point of view is supported by many numerical results (see [21], for instance), and by the ill-conditioning problems that occur when one tries to calculate the required vector of variables to high accuracy. Some interesting and useful algorithms can be obtained, however, by including Lagrange parameter estimates in the definition of the penalty function. For example, we consider briefly the augmented Lagrangian method [11], in the case when the only constraints on the variables are the equations

$$c_i(\underline{x}) = 0, \quad i = 1,2,\ldots,m . \tag{3.1}$$

The method employs an algorithm for unconstrained optimization to calculate the vector $\underline{x}$ that minimizes the function

$$L(\underline{x},\underline{\lambda},r) = F(\underline{x}) - \sum_{i=1}^{m} \lambda_i c_i(\underline{x}) + r \sum_{i=1}^{m} [c_i(\underline{x})]^2, \tag{3.2}$$

for several values of the parameters $\underline{\lambda}$ and r. We let $\underline{x}(\underline{\lambda},r)$ be the calculated vector of variables, and, assuming that the optimization problem is not pathological, there exist parameters $\underline{\lambda}^*$ such that the Lagrangian function

$$F(\underline{x}) - \sum_{i=1}^{m} \lambda_i^* c_i(\underline{x}) \tag{3.3}$$

is stationary at the required solution, $\underline{x}^*$ say. Therefore, for all values of r, the function (3.2) is also stationary at $\underline{x} = \underline{x}^*$ if $\underline{\lambda} = \underline{\lambda}^*$. Increasing the value of r usually forces the stationary point to be a minimum, and in the augmented Lagrangian method it is necessary for r to reach a value that makes $\underline{x}(\underline{\lambda}^*,r)$ equal to $\underline{x}^*$. Then r is fixed, and, by using the calculated vectors $\underline{x}(\underline{\lambda},r)$, the parameters $\underline{\lambda}$ are adjusted automatically so that they converge to $\underline{\lambda}^*$. Hence $\underline{x}(\underline{\lambda},r)$ converges to the required solution. Three good properties of the method are

ethods to revise second derivative estimates, because they provide a atural way of preserving sparsity. A successful algorithm of this kind as been found that is based on the Frobenius matrix norm (see [22], or instance), but it is more difficult to develop an algorithm that is imilar to the BFGS method [19]. Another point to keep in mind in this ield of research is that the sparsity can make it highly efficient to btain second derivative approximations from differences in first deriative vectors [20].

lgorithms for unconstrained minimization that do not require the calcuation of derivatives are also important. It seems to me, however, that o useful progress has been made in this subject for many years. It is sual to estimate derivatives by differences in order to apply a gradient thod, and often the resultant algorithm is the most efficient of the nes that are available. In this case, if n is large, then nearly all unction calculations are for the purpose of derivative estimation, instead of being used directly for the main task of reducing the objective unction, which does not seem to be a sensible balance of effort.

en there are linear constraints on the variables, then it is usual to pply an active set strategy as in quadratic programming (see [10] for xample), which reduces the calculation to a sequence of problems in ich all the constraints are equalities. Because these constraints can used to eliminate variables, one really has a sequence of unconstrained roblems. If the BFGS algorithm is applied to each one, then useful secd derivative information can be passed from one problem to the next 16], but unfortunately it seems that the conjugate gradient algorithm s not have this property. However, each unconstrained calculation is ften small because the number of variables is n *minus* the number of tive constraints. Moreover, it is possible to apply the techniques that used in linear programming to take advantage of any sparsity in the fficients of the constraints. Therefore, when the number of active nstraints is close to n , it is possible to solve problems in which number of variables is very large. The algorithm that is described y Murtagh and Saunders [15] is recommended, but some further work may needed on the rules that govern the removal of a constraint from the ctive set.

is technique for linear constraints can be extended to the general nonlinear optimization problem, provided that one can use nonlinear equality nstraints to eliminate variables. The resultant algorithm is called the

* * OTHER COAL-RELATED TESTING * *

TESTING MINIMIZATION CODES

A. Buckley
Concordia University, Loyola Campus
Department of Mathematics
714 Sherbrook Street, West
Montreal, Quebec, H4B 1R6, Canada

1. Introduction. I have been asked to briefly describe some ideas for testing minimization algorithms which are currently being implemented. The aim here is not to perform and describe results of extensive software evaluation. Rather the intention is to provide a tool to assist people, particularly those who design algorithms, to test codes thoroughly. I hope that these programs will significantly reduce the effort required to perform such testing, and that they will provide a consistent framework for testing which will facilitate comparison of results.

There are three principal aims which have guided the design of these routines. First, they must be as convenient as possible to use, but without being inflexible. Thus, the input control procedures have been carefully designed and are quite powerful. Second, they must provide a reasonable set of performance statistics. A lot of effort has gone into the formatting of the output to make it clean, readable, and attractive. Finally, they must be flexible so that any algorithm can be tested.

The testing package essentially consists of four parts. First, many "standard" test functions are included. Amongst these are all those from the Argonne package. Second, a main program is used to control the testing procedure. Any minimization algorithm can be implemented, and the program will test the given algorithm on any group of problems selected by the user and produce a summary. The third part is a collection of routines to perform chores occurring in many minimization codes. These are summarized later, but with them the programmer of a minimization code may concentrate on his algorithm without the need to worry about inessential details. They also contribute to uniformity in testing. The final part of the package is a file describing a collection of standard test problems. All the information needed to run a test, such as starting point, dimension, name of test function, etc., is included.

Before I forget, the code is written in FORTRAN since it is most widely used in North America. It is written to be PORTABLE; otherwise, there would be no point to this exercise. But I must add that it is not written using the popular 1966 standard. In order to produce programs of tolerable structure and readability, the new ANSI 1977 Standard FORTRAN was used.

2. The Test Functions. I have provided codes for many standard test problems, each with a standard calling sequence. For example, the ubiquitous function introduced by Rosenbrock appears as SUBROUTINE ROSENB(N,X,F,G,IFL). The only parameter requiring comment is IFL. If IFL $\geq$ 0 , then the function value F at X is determined; if IFL $\leq$ 0 , then the gradient G at X is computed. Clearly, IFL = 0 gets both.

3. The problem file. Nine typical lines of the problem file are as follows:

```
*ROSENBRK  THIS IS THE EVER-FAMOUS ROSENBROCK FUNCTION.
0001 0001 0002 00100
-1.2       1.0
*ROSNBRK2  THIS IS ROSENBROCK'S FUNCTION WITH A DIFFERENT XO.
0002 0001 0002 00100
2.5        -1.6
*HELIX     THIS IS THE HELICAL VALLEY FUNCTION, COMMONLY CALLED HELIX.
0012 0013 0003 00100
-1.0       0.0        0.0
```

Here we have 3 standard test problems specified. The first is named ROSENBRK. The line "THIS IS ..." will appear in the user's output for identification. The data, in order, indicates: This is problem number 1, the test function is number 1, the dimension is 2, at most 100 function evaluations should be required and the starting point is -1.2, 1.0. The next 3 lines specify ROSNBRK2 as problem #2 using the same function #1 of dimension 2 but starting at 2.5, -1.6. To choose a test problem you simply specify either its name, e.g. HELIX, or its number, e.g. 12. The problem is then found and the data is read.

4. The subsidiary routines. I will briefly describe the main function of each of these. They do more than these brief descriptions indicate.

(i) EVALF (FUNCT,N,X,F,G,IFL): This calls FUNCT(N,X,F,G,IFL) (see §2.) The time taken for the call is recorded, the number of function and gradient calls are separately counted and termination is forced when the function count reaches a preset maximum. Scaling of the function and tracing are available within EVALF.

(ii) PRINT (N,X,F,G,IFORCE): This prints F and the function call, gradient call and iteration counts at specified iterations, say every kth. Optionally, the X and G vectors may also be printed. PRINT determines when output is required, unless IFORCE = 1 , which forces printing, say when the solution is found.

(iii) HSTOP (N,VEC1, VEC2, EPS, LESS): This provides a means of applying a uniform stopping criterion for different algorithms. Depending on parameters preset in COMMON, different tests are available. For example, setting VEC1 = G , one may test if the gradient G satisfies $\| G \|_p \leq$ EPS, or setting VEC1 = x_i and VEC2 = x_{i+1} , one may test if $\| x_i - x_{i+1} \|_p \leq$ EPS . One may choose p = 1,2 or ∞ . Of course the stopping test could be written as part of the algorithm being tested. But, using HSTOP makes it easy to change the stopping criterion to provide results which are consistent with published results for another algorithm using a different stopping test.

These three routines simplify programming; you can ignore details such as counting function calls or producing output. They are also quite flexible (for these descriptions are rather incomplete), and they also provide some uniformity in implementing different algorithms.

3. A review of algorithms for optimization

Because of the delays that have just been mentioned, a review of publishe papers and of current work gives a much better idea of the state of the art of optimization than a study of computer subroutines. Therefore this section surveys briefly the main areas of optimization, and it includes some comments on recent research.

For unconstrained optimization calculations, when first derivatives can be calculated and when an approximate second derivative matrix can be stored, the BFGS algorithm (see [9], for instance) is so reliable and so efficient that it is the first choice of many computer users. It is the method that I prefer, unless the objective function has some useful stru ture. Among the other algorithms for this calculation, there are two that were proposed by Davidon that are particularly interesting. The first of them [6] has the highly attractive property of giving quadratic termination, without line searches and without the use of any indefinite matrice One purpose of the second algorithm [7] is to take account of both function and gradient information from previous iterations, because the BFGS method uses only changes in gradient vectors. However, I know of no stron numerical evidence that suggests that either algorithm is better than th BFGS algorithm in practice, but little experience has been obtained so f with Davidon's newest technique.

When matrices cannot be stored, the conjugate gradient algorithm is often an excellent one to use. The version that is in the IMSL library includes an automatic restarting procedure that usually reduces the total number of iterations [17]. One promising new idea in this field is to combine the BFGS and conjugate gradient methods in a way that can take full advantage of a limited amount of computer storage [3]. This work may help to improve the efficiency of the conjugate gradient algorithm in the case when there are linear inequality constraints on the values of the variables.

Much attention has been given recently to methods that take advantage of any sparsity in the elements of the second derivative matrix of the objective function of an unconstrained minimization calculation. They are important because, if the sparsity pattern is retained in the approximations to the second derivative matrix, then it is often possible to apply matrix methods to calculations that have many more variables than before. Most of the work so far has been concerned with the use of variational

the progress of a new idea, perhaps for two or three years. If an algorithm passes this test, then one tries to include the algorithm in the plans that have been made for the addition of new routines. Further delays are likely to occur, unless the new algorithm is in a field of optimization that is the subject of planned future work. When the time comes to include the new technique in the library, then it is sometimes easiest to obtain a computer listing from the author, and to make changes to the listing only if they are essential. Often, however, the easiest course is not followed. One reason is that one may prefer to adapt the new routine to the existing structure of library routines, especially if there is a policy of composing the library of small modules, which was once the scheme of MINPACK [2]. Another reason for not accepting the author's listing is that, although one wishes to apply his main method, one may object to some important details, such as a procedure that is used for line searches, or a condition for convergence. Therefore a completely new computer program may be written, that is very different from the one that was developed by the author of the algorithm. If the reprogramming also gives attention to robustness in the way that is advocated by Moré [14], if the new program is suitable for many different computing machines, and if the library requires all subroutines to be validated independently before they are made available generally, then the effort that is needed to include a subroutine in the library is very great. It is usually well worthwhile, because it makes it possible for thousands of computer users to apply an algorithm to their research, but libraries generally will remain at least five years behind the development of new techniques, except when authors of algorithms contribute directly to a library.

5. Running tests. Tests may be run from batch, or from a terminal. Minimal input is required; ease of use is foremost. The input routines are quite powerful; if the input makes sense to you it is likely acceptable to the machine. We illustrate here with part of a sample terminal session. (The "?" is a prompt by our CDC NOS system.)

	TERMINAL I/O	COMMENTS
(1)	ENTER TITLE LINE ... UP TO 78 CHARACTERS ? This is to illustrate the testing package.	Identification for the output (see §6).
(2)	ENTER CONTROL PARAMETERS ? NORM=2, GRADIENT, ACC 1.E-5, PRINT5	Control execution. NORM, GRADIENT and ACC apply to HSTOP (§4.iii); printing is every 5th iteration (§4.ii).
* (3)	WHICH METHOD? ? BFGS	Not normal input (see §7).
(4)	WHICH PROBLEMS? ? HELIX 1 OREN20 END	Test using 3 problems: HELIX(#12), #1(ROSENBRK) and OREN20(#45). The order doesn't matter.

Note that the line "NORM=..." could be typed equivalently as " NORM= EUCLIDEAN , TEST=GRADIENT, ACCURACY=1.E-5, PRINT=5" or at the other extreme as "EUCL GRAD A1.E-6 P5". If at step (4), you change your mind about the accuracy, or find an earlier typing error, just type "GOTO2" instead of "END" and you'll get a chance to make the correction. Parameters not assigned take default values of course. If you want the same values as the last test, try "REPEAT". If you forget what is required, type "HELP". Punctuation may be omitted if the meaning is clear. Keywords can be abbreviated or omitted.

6. Output. A small compressed sample of output is provided. Only the line preceded by * was output by the algorithm being tested. The rest was provided by routines in the package. A marginal "←n" notes where n lines of output were deleted. Note the final summary. It should be self-explanatory, except that MSEC is the total time the algorithm took in the minimization of the function, and FSEC is the time taken in evaluating the function. Print time is automatically excluded.

7. User control. I have claimed that this package may be used to test any unconstrained minimization algorithm. That is so. Of course one must be able to link any routine to the test package, and facility is made for doing this easily. Also, every minimization algorithm has its own personal list of special parameters and its own idiosyncracies, and care has been taken to allow inclusion of such special features without requiring any major revisions to the code. For example, in the tests illustrated (see §5 and §6) the author used an algorithm implementing 2 methods (Shanno's CG routine from TOMS). The line marked * was added as special input and was to determine which method should be used for this test.

8. Conclusion. It is hoped that others will find these programs useful in algorithm testing. Ease of use and thoughtful output should make it attractive to use. It is currently running but improvements are being made. Suggestions would be most welcome. It should be mentioned that it is very easy to modify existing algorithms to run with this package. For example, I implemented VA08 and VA14 from the Harwell library and Shanno's CG algorithm with inexact line searches with no difficulty at all.

```
 THIS IS TO ILLUSTRATE THE TESTING PACKAGE.
 TEST BEING EXECUTED AT  9:30 A.M., SEPTEMBER 11, 1980

 THIS IS THE HELICAL VALLEY FUNCTION, OFTEN CALLED HELIX.

 STANDARD CONTROL PARAMETERS:
      TERMINATION NORM =  2 (EUCLIDEAN)
      TERMINATION TYPE =  1 ( GRADIENT)
      ACCURACY SPECIFIED          .100E-04

 ADDITIONAL USER DEFINED CONTROLS:
 NMETH =  1

 .....HAVING REACHED THE POINT NUMBER     0 .....
 THE FUNCTION VALUE IS    .104000000E+03 (    1 FUNCTION EVALUATIONS).
                                         (    1 GRADIENT EVALUATIONS).
 THE VARIABLES HAVE THE CURRENT VALUES GIVEN BY
  -.10000000E+01 0.              0.
 THE GRADIENT AT THIS POINT IS
  -.40400000E+03 -.20000000E+03 0.

 .....HAVING REACHED THE POINT NUMBER     5 .....
 .....HAVING REACHED THE POINT NUMBER    10 .....+7
 .....HAVING REACHED THE POINT NUMBER    15 .....+7
 .....HAVING REACHED THE POINT NUMBER    20 .....+7
 .....HAVING REACHED THE POINT NUMBER    25 .....+7
                                                 +6
 THE SOLUTION HAS BEEN FOUND:

 .....HAVING REACHED THE POINT NUMBER    27 .....
 THE FUNCTION VALUE IS    .523089694E-15 (   38 FUNCTION EVALUATIONS).
                                         (   38 GRADIENT EVALUATIONS).
 THE VARIABLES HAVE THE CURRENT VALUES GIVEN BY
   .10000000E+01  .10000000E+01 0.
 THE GRADIENT AT THIS POINT IS
   .27978996E-06 -.11779520E-06 0.
                                                 +1
    SUMMARY OF PROBLEM:
     PR# FN#   NAME     DIM ITS FNCS GRDS  FVALUE  GVALUE MSECS FSECS ER
      12  13 HELIX        3  27   38   38 .52E-15 .30E-06   .17   .01  0
                                                 +1
                                                +450
 HERE IS A COPY OF THE SUMMARY DATA:             +1
 THIS IS TO ILLUSTRATE THE TESTING PACKAGE.
 TEST BEING EXECUTED AT  9:30 A.M., SEPTEMBER 11, 1980
 STANDARD CONTROL PARAMETERS:
      TERMINATION NORM =  2 (EUCLIDEAN)
      TERMINATION TYPE =  1 ( GRADIENT)
      ACCURACY SPECIFIED          .100E-04

 SUMMARY OF PROBLEMS DONE...

  PR# FN#   NAME     DIM ITS FNCS GRDS  FVALUE  GVALUE MSECS FSECS ER
   12  13 HELIX        3  27   38   38 .52E-15 .30E-06   .17   .01  0
    1   1 ROSENBRK     2  36   44   44 .37E-13 .68E-05   .19   .01  0
   28   5 OREN20      20 100  100  100 .19E-02 .70E-01  5.19   .13  1
                     ITERS FUNCS GRADS                 MSECS  FSECS
 TOTALS                163   182   182                 5.55    .16

 THERE WERE    1 PROBLEMS FLAGGED WITH ERRORS.
```


lems can be solved, however, because there are several recent procedures that apply the augmented Lagrangian method. The total number of optimization routines is already quite large, partly because, if a gradient meth is included, then there is usually a version of the method that uses differences to avoid the analytic calculation of first derivatives. It takes far more work to develop a NAG routine than a Harwell routine, due to the NAG policy of providing their routines in several languages for several types of computer.

Although the IMSL library is excellent in many fields of numerical and statistical calculations, it is very weak in optimization. One reason is that the library authorities preferred to exclude routines that require the user to provide derivatives. In my opinion this policy is detrimental to algorithm development because, if a new method is basically a gradient method, it is usually best to gain experience with the algorithm in its true form, before providing a version that makes difference approximations to derivatives. Moreover, the calculation of analytic derivatives can save computer time and can gain accuracy. I intend to provide IMSL with some good optimization routines, because I am now one of their consultants.

This review suggests that there are likely to be long delays in making new optimization algorithms available to computer users as library routines, unless the library has a flexible policy that makes it easy for the authors of new algorithms to contribute their work. Harwell can maintain such a policy, because the close contact between the chief librarian and the users for whom the library is intended allows any corrections and modifications to routines to be made quite rapidly. The standards and policies of NAG, however, make it more difficult to include new routines, which is one of the reasons why the NAG library does not yet include an algorithm for minimizing a general objective function subject to general linear constraints. This calculation occurs frequently in practice, and a successful method for its solution was proposed by Goldfarb in 1969 [12]. The following discussion suggests that some delays of this kind may be unavoidable.

Suppose that one is responsible for the optimization section of a library that has the distribution and policies of NAG, and that quite unexpectedly one learns of a new algorithm that may provide substantial improvements over existing techniques. If one believes the claims of authors, then this situation occurs so often that it is not possible to study carefully each promising new method. Therefore one usually waits and watches

2. Subroutine libraries

Many general programs for numerical calculations are available in subroutine libraries, and many important optimization problems are solved now by library procedures. One of the main advantages of a library routine is that there is usually no need for the user to understand in detail how the algorithm works. Also the use of library routines can reduce greatly the amount of new computer code that has to be written to carry out a numerical calculation. Therefore this section considers the range of optimization algorithms that are available as library procedures. Partly because of my limited experience only the Harwell, NAG and IMSL libraries are discussed. I believe that at present these three libraries are used more than any others as sources of general routines for numerical calculations.

A fundamental difference between the Harwell library and the other two is that the main purpose of the Harwell library is to provide computer users at only one establishment with general routines, while NAG and IMSL are both designed to be suitable for a wide range of computer installations. Therefore routines that are coded in Fortran IV for an IBM 360 or 370 computer are acceptable to Harwell, which makes it easy for the staff and visitors at Harwell to contribute the algorithms that they develop in the course of their research. Nearly all the Harwell optimization routines were collected in this way. They include several routines for unconstrained and linearly constrained optimization calculations, an augmented Lagrangian method for nonlinear constraints that was provided by Fletcher in 1974, and a variable metric method for nonlinear constraints that I contributed in 1977. There is a catalogue that gives a list of these routines [13]. Copies of the Harwell library have been obtained by several hundred computer installations throughout the world. Because of its relevance to the discussion of Section 4, we note that many of these installations do not have an IBM computer.

The high quality of the optimization section of the NAG library is due largely to the work of Gill and Murray. They are covering the different areas of optimization carefully and methodically. One area that is particularly strong is algorithms for unconstrained optimization, in the case when full matrices can be stored, and it includes routines for nonlinear least squares calculations. Some of these programs have been extended to allow simple bounds on the variables, but there are not yet any routines for general linear constraints. More general constrained prob-

NOTE ON SMALL MODEL WHICH CYCLED

C. A. Haverly
Haverly Systems, Inc.
78 Broadway, P. O. Box 919
Denville, New Jersey 07834

Jan Telgen reported in the COAL Newsletter (1) on his experience with the small model of Kotiah and Steinberg (2) which cycled.

The model is 16 rows by 20 structural columns. It has all zero right hand side values except one and the matrix coefficients are poorly scaled.

I tried the model on our new HS/LP system for mini computers (run on a Data General S130). The model solved in 22 iterations with no difficulty using the default settings of parameters. The code selected to price the entire matrix and to choose the four best candidates on each pricing pass.

In order to explore the effect of the solution strategy on the cycling, I first set the number of candidates to one. This was much less effective and 90 iterations were required to solve the model. Eighty-five of these iterations were involved in getting feasible. The model entered a 9 iteration cycle at iteration 31 and one infeasibility. The vectors entering the basis were, in order: 12, 17, 13, 18, 9, 1, 11, 8, 9, 12 . . . , . An automatic invert at iteration 41 did not effect the cycle. But an automatic invert at iteration 81 (a different point in the cycle) broke the cycle and the model quickly became feasible and optimal.

The model was then run with the one best candidate selected after pricing only part of the matrix (priced until 7 possible candidates found). The model did not cycle but took 44 iterations to solve.

When the dynamic pricing was changed from 7 to 5 (still with a single best candidate selected), the model began, at iteration 31 the same cycle as before. An automatic invert again broke the cycle and the model was solved in 90 iterations.

When the preceeding strategy was used with right hand side perturbation, the cycle did not form and the model solved in 36 iterations.

Full matrix pricing and selection of the two best candidates per pass resulted in reaching optimal in 21 iterations without any cycling. When perturbation was added, we found that 36 iterations were required. When perturbation was used with full matrix pricing and selection on the one best candidate, we did not get any cycling as we did without perturbation. The model solved in 44 iterations.

Full matrix pricing and selection of the two best candidates per pass resulted in reaching optimal in 21 iterations without any cycling. When perturbation was added, we found that 36 iterations were required. When perturbation was used with full matrix pricing and selection on the one best candidate, we did not get any cycling as we did without perturbation. The model solved in 44 iterations.

CONCLUSIONS

Multipricing, partial matrix pricing, inversion, and perturbation all have the possibility of breaking cycles.

(1) Telgen, Jan, "A Note on a Linear Programming Problem that Cycled," COAL Newsletter, August 1980, pp. 8-11.

(2) Kotiah, T. C. T. and Steinberg, D. I., "Occurrence of Cycling and Other Phenomena Arising in a Class of Linear Programming Models," Communications of the ACM, 20, No. 2 (1977), pp. 107-112.

that are required to specify constraint information, and also because of the extra storage that is needed for the section of the computer program that takes account of the constraints. The techniques for constrained optimization depend on whether or not the constraint functions are linear. In the linear case it is possible to keep the trial vectors of variables feasible, except for computer rounding errors, during most of the calculation, but, if there is a nonlinear equality constraint, then usually feasibility can be obtained only by an iterative correction procedure. Two other important considerations are whether the number of variables is so large that it is not possible to store full nxn matrices, and whether there is any useful structure in the program, for example the objective function may be a sum of squares, or it may be known that many elements of the second derivative matrix of $F(\underline{x})$ are zero. These two considerations are different. Sparsity can be used sometimes to reduce the amount of calculation even when n is small, and the conjugate gradient method shows that sparsity is not the only key to the solution of large problems. Moreover, the choice of algorithm may depend on whether derivatives can be calculated.

Published methods solve only a few of these types of optimization calculations. One way of reviewing them is to consider the computer programs for optimization that are present in subroutine libraries. This point of view is taken in Section 2, but we find that many useful algorithms are not yet available as general computer programs. The reasons for the delay are discussed, and it seems to be inevitable that general subroutine libraries are about five years behind the development of new methods.

Therefore Section 3 mentions some of the techniques for optimization that have been published recently. Particular attention is given to the solution of large unconstrained optimization problems, and to variable metric methods for nonlinear constraints. Some numerical results show that the variable metric methods are more efficient than earlier algorithms if efficiency is measured by the number of times the objective and constrain functions and their gradients are evaluated during the solution of each optimization problem.

I provided a Fortran subroutine about two years ago, for solving the constrained optimization problem by a variable metric method. Its successes and failures are discussed in Section 4, because they show some of the difficulties that can occur when one makes available a computer program for a new algorithm. The discussion suggests that, in addition to the present high standards of mathematical software, there is a need for a more tolerant standard.

****FEATURE ARTICLE****

OPTIMIZATION ALGORITHMS IN 1979*

M.J.D. Powell
Department of Applied Mathematics and Theoretical Physics,
University of Cambridge,
England

1. Introduction

Many applications of optimization techniques give the following mathematical problem. Calculate a vector of variables $\underline{x}$, that minimizes an objective function $F(\underline{x})$, subject to the conditions

$$c_i(\underline{x}) = 0, \quad i = 1,2,\ldots,m', \tag{1.1}$$

and

$$c_i(\underline{x}) \geq 0, \quad i = m'+1,\ldots,m, \tag{1.2}$$

on the values of the variables. We reserve n for the number of components of $\underline{x}$. We suppose that all functions are real, and that their values can be calculated for any $\underline{x}$.

The purpose of this paper is to review briefly the general algorithms that are available for solving the optimization problem that has just been described. Because the review is restricted to my own field of interest, I will discuss only local methods that depend on the existence of derivatives. In a local method, each change to an estimate of the solution depends on the behaviour of the objective and constraint functions in a neighbourhood of the estimate. Therefore we have to take the point of view that $\underline{x}^*$ is an acceptable solution to the main calculation, if it satisfies the constraints, and if any small change to $\underline{x}^*$ that preserves feasibility causes the value of the objective function to increase. Although differentiability is assumed, I will comment briefly on some algorithms that do not require derivatives to be calculated.

Even under these restrictions, the range of relevant algorithms is very wide indeed. If one could restrict attention to the best algorithm for each of the main types of optimization calculations, then the range would still be large, because of the number of factors that are relevant to the choice of a suitable algorithm. For example, if there are no constraints on the values of the variables, then one is unlikely to choose an algorithm that can treat constraints, partly because of the extra arguments

*This article is reprinted from "Lecture Notes in Control and Information Sciences" Volume 22, published by Springer-Verlag, Berlin.

Calendar of mathematical programming meetings
as of 1 January 1981

Maintained by the Mathematical Programming Society (MPS)

This Calendar lists meetings specializing in mathematical programming or one of its subfields in the general area of optimization and applications, whether or not the Society is involved in the meeting. (These meetings are not necessarily "open".) Any one knowing of a forthcoming meeting not listed here is urged to inform the Vice Chairman of the Society, Dr. Philip Wolfe, IBM Research 33-221, POB 218, Yorktown Heights, NY 10598, U.S.A; Telephone 914-945-1642, Telex 137456.

Some of these meetings are sponsored by the Society as part of its world-wide support of activity in mathematical programming. Under certain guidelines the Society can offer publicity, mailing lists and labels, and the loan of money to the organizers of a qualified meeting. For further information address the Chairman of the Executive Committee, Dr. A. C. Williams, Computer Science Department, Mobil Oil Co. Technical Center, Box 1025, Princeton, New Jersey 08540, U.S.A.; Telephone 609-737-3000, extension 4342.

Substantial portions of regular meetings of other societies such as SIAM, TIMS, and the many national OR societies are devoted to mathematical programming, and their schedules should be consulted.

1981

January 5-6: "Mathematical Programming: Testing and Validating Algorithms and Software". U. S. National Bureau of Standards, Boulder, Colorado. Organized by the Committee on Algorithms of the MPS, the Bureau of Standards, and the Department of Energy. Contact: Dr. Richard H. F. Jackson, Center for Applied Mathematics, National Bureau of Standards, Washington, D.C. 20234, U.S.A.; telephone 301-921-3855.

January 26-31: "Mathematische Optimierung", Mathematisches Forschungsinstitut Oberwolfach, Oberwolfach, Federal Republic of Germany. Contact: Institut für Ökonometrie und Operations Research (see 1982, August 23-28).

March 16-20: "Optimization: Theory & Algorithms", Confolant (Miremont, Puy-de-Dôme) France. Contact: Professor J.-B. Hirriart-Urruty, Département de Mathématiques Appliquées, Université de Clermont-Ferrand II, B.P. 45, 63170 Aubière; Telephone (73) 26-41-10.

April 6-8: "International Congress on Mathematical Programming", Rio de Janeiro, Brazil. (Abstract deadline 1 December 1980; special forms required) Contact: Professor Milton Kelmanson, Caixa Postal 1507 - CEP 20100, Rio de Janeiro, R.J., Brazil. Sponsored by Sociedade Brasiliera de Pesquisa Operacional and the MPS.

May 13-14: "Optimization Days", Université du Quebéc à Montréal. Contact: Professor Efim Galperin, Département de mathématiques, Université du Québec à Montréal, C.P. 8888 Succ. "A", Montréal, Québec, Canada H3C 3P8; telephone 514-282-3221. Sponsored by IEEE and the MPS.

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.

July 20-24: "Eighth British Combinatorial Conference", Swansea, England. Contact: A.D. Keedwell, Department of Mathematics, University of Surrey, Guildford, Surrey GU2 5XH, U.K.

July: "Stochastic Programming", Budapest, Hungary. Contact: Bolyai János Mathematical Society, Budapest VI, Anker köv 1-3, I. Em. III, Hungary.

August 24-28: "CO81: Conference on Combinatorial Optimization", Stirling, Scotland. Contact: Professor L. Wilson (CO81), Department of Computing, Stirling University, Scotland, U.K.

1982

August 23-28: Eleventh International Symposium on Mathematical Programming in Bonn, Federal Republic of Germany. Contact: Institut für Ökonometrie und Operations Research Universität Bonn, Nassestraße 2, 5300 Bonn 1, Federal Republic of Germany; Telex 886657 unibo b, Telephone (02221) 739285. Official triennial meeting of the MPS. (Note: The International Congress of Mathematicians will be held August 11-19 in Warsaw, Poland.)

****

THE MATHEMATICAL PROGRAMMING SOCIETY

ENROLLMENT

I hereby enroll as a member of the Society for the calendar year 1981.

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 1981 are:

42 Dollars (U.S.A.)
17.50 Pounds (U.K.)
69 Francs (Switzerland)
176 Francs (France)
76 Marks (Fed. Rep. Germany)
82 Guilders (Netherlands)

Please send this application with your dues to

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

EDITOR'S COLUMN

The feature article of this newsletter is one by Professor M. J. D. Powell about the current publication standards for mathematical software. This article was previously published in Issue 22 of "Lecture Notes in Control and Information Sciences" and has been reproduced in its entirety in this newsletter because I believe it presents a viewpoint contrary to that normally presented in this newsletter. I encourage each of you, after reading this article, to write to me indicating your views on this subject.

This article is especially timely since COAL is reviewing all of its current tasks and considering what future directions the committee should take. Should COAL be primarily concerned with disseminating information about codes, test problems, and testing? Should it concentrate on performing major test efforts itself? Should it develop and refine guidelines for reporting computational testing of MP software? These are, of course, not mutually exclusive alternatives, and we welcome suggestions from our readers on what would be of most benefit.

Articles in this newsletter point to the fact that there has been an increase in test efforts in the recent past. The program for the Conference on Software and Testing, sponsored by COAL in Boulder on January 5-6, 1981, also highlights this recent increase in testing. (The August 1980 issue of this newsletter published this program.) Proceedings of that conference will be forthcoming. If you would like to obtain a copy of that publication, you should contact John Mulvey, School of Engineering/Applied Science, Princeton University, Princeton, New Jersey 98540.

Finally, I would like to apologize to all the authors of articles in this issue. They rushed to provide me with their input before the Christmas holiday since I promised to have the newsletter out by the end of the year. Unfortunately, an illness forced the later publication. I hope to be more punctual in the future.

Sincerely,
KARLA L. HOFFMAN

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.

DISTRIBUTION OF THIS NEWSLETTER

This newsletter is mailed to every member of the Mathematical Programming Society and to all "friends" of COAL. If you are not presently receiving this newsletter and would like to, please write to the editor requesting that your name be added to the list of "friends" of COAL. There is currently no charge for this newsletter.

Mathematical Progamming Society
Committee on Algorithms Newsletter

No. 5: February 1981 Karla L. Hoffman, Editor

Contents:

DR. KARLA L. HOFFMAN
UNITED STATES DEPARTMENT OF COMMERCE
NATIONAL BUREAU OF STANDARDS
CENTER FOR APPLIED MATHEMATICS
WASHINGTON, D.C. 20234

PRINCE GEORGES, MD
PM
FEB 25
1981
207

T. Steihaug
School of Organiza. & Mgmt
56 Hillhouse Ave.
New Haven, Ct 06520
USA

THIRD CLASS MAIL