REGION COMPARATIVE TESTING

-29-
This means for the implementation that we shall define an algorithm to be convergent if both the following criteria are satisfied:

\[ \|x_k - x_{k-1}\| \leq \epsilon_1(\|x_k\| + 1) \]

for some pre-given precision parameter \( \epsilon_1 > 0 \)

and

\[ |c_i(x_k)| \leq \epsilon_2(\|x_k\| + 1), \text{ for all currently violated} \]

constraints and for some precision parameter \( \epsilon_2 > 0 \)

The precision parameters used are \( \epsilon_1 = \epsilon_2 = 10^{-5} \).

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 \( c = 0.01 \) or if the distance of the successively generated iteration points along the line is smaller than or equal to \( \epsilon_h \), with \( \epsilon_h = 10^{-2} \).

This last criterion is also applied in the line search of the 2-phase algorithms, with \( \epsilon_h = 10^{-4} \).

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 \( VF(x_k) \), is small enough: \( \|p_k\| \leq \epsilon_5 \), with \( \epsilon_5 = 10^{-4} \) and at the same time the expected improvement if a constraint is dropped is less than \( \epsilon_5 \).

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
Recent Computational Testing

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 time'. 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 Staeh (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 \( x^* = (1, \ldots, 1) \). 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 \( e_i = 10^{-6}(|x_i| + 0.001) \) 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

<table>
<thead>
<tr>
<th>Sex</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Female</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

**Table 2:**

**Number of subjects**

1. The experiment was conducted in a controlled environment with constant temperature and lighting conditions.
2. Subjects were randomly assigned to either the experimental group or the control group.
3. The experimental group received a specific treatment, while the control group did not.
4. Data collection took place over a period of four weeks.
5. Data was analyzed using statistical software.

**Discussion of the experimental results**

The results indicated that the experimental treatment had a significant impact on the outcome, as compared to the control group. Further studies are recommended to validate these findings.
at every major iteration of phase II, is more robust than HQF. 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 the active set but once it obtains it, 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 ends up with the same active set.

When comparing our results with those in Staha (1973), we should be aware of some small 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 1900 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. But it still seems to be justified to state that algorithms I, II and III, behave very well in comparison with COMET, Lootsma and GPMRLC. 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 diagnostic work was required to prevent failures for GREG. He reports that sometimes an artificial constraint, such as \( \sum_{i=1}^{n} x_i \leq 10 \times 10^{10} \) 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 \( 1 + (u - |N|/2)^2 \) 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 I reflect the average solution time in CPU seconds over all problems of the indicated dimensions.

<table>
<thead>
<tr>
<th>Topology</th>
<th>NODES</th>
<th>ARCS</th>
<th>HQF</th>
<th>MAXAUG</th>
<th>SUBRED</th>
<th>PRIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure</td>
<td>250</td>
<td>1250</td>
<td>0.13</td>
<td>0.10</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>Random</td>
<td>250</td>
<td>1675</td>
<td>0.24</td>
<td>0.18</td>
<td>0.16</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>2500</td>
<td>0.29</td>
<td>0.25</td>
<td>0.21</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>2500</td>
<td>0.32</td>
<td>0.32</td>
<td>0.23</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>3750</td>
<td>0.55</td>
<td>0.51</td>
<td>0.49</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>5000</td>
<td>0.60</td>
<td>0.51</td>
<td>0.50</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>3750</td>
<td>0.46</td>
<td>0.42</td>
<td>0.36</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>5825</td>
<td>0.41</td>
<td>0.35</td>
<td>0.40</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>7500</td>
<td>0.77</td>
<td>0.71</td>
<td>0.65</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>5000</td>
<td>0.43</td>
<td>0.38</td>
<td>0.43</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>7500</td>
<td>0.84</td>
<td>0.79</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>10000</td>
<td>1.01</td>
<td>0.98</td>
<td>1.21</td>
<td>1.56</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multi-terminal</th>
<th>NODES</th>
<th>ARCS</th>
<th>HQF</th>
<th>MAXAUG</th>
<th>SUBRED</th>
<th>PRIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>250</td>
<td>1250</td>
<td>0.53</td>
<td>0.39</td>
<td>0.18</td>
<td>0.29</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>1875</td>
<td>2.76</td>
<td>1.14</td>
<td>0.37</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>2500</td>
<td>2.51</td>
<td>1.74</td>
<td>0.36</td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>2500</td>
<td>0.97</td>
<td>0.72</td>
<td>0.29</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>3750</td>
<td>2.00</td>
<td>1.42</td>
<td>0.40</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>5000</td>
<td>0.49</td>
<td>0.63</td>
<td>0.60</td>
<td>1.63</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>3750</td>
<td>0.85</td>
<td>0.85</td>
<td>0.48</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>5825</td>
<td>2.58</td>
<td>2.58</td>
<td>0.65</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td>750</td>
<td>7500</td>
<td>6.50</td>
<td>6.50</td>
<td>1.00</td>
<td>2.75</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>5000</td>
<td>0.70</td>
<td>0.51</td>
<td>0.87</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>7500</td>
<td>1.09</td>
<td>0.78</td>
<td>2.19</td>
<td>2.19</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>10000</td>
<td>1.02</td>
<td>1.25</td>
<td>5.67</td>
<td>5.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dense</th>
<th>NODES</th>
<th>ARCS</th>
<th>HQF</th>
<th>MAXAUG</th>
<th>SUBRED</th>
<th>PRIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>acyclic</td>
<td>20</td>
<td>190</td>
<td>0.20</td>
<td>0.17</td>
<td>0.17</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>780</td>
<td>3.73</td>
<td>1.29</td>
<td>1.29</td>
<td>0.47</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>1170</td>
<td>4.24</td>
<td>1.95</td>
<td>1.95</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>110</td>
<td>2120</td>
<td>11.65</td>
<td>10.65</td>
<td>11.65</td>
<td>3.31</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>2505</td>
<td>15.85</td>
<td>13.85</td>
<td>13.85</td>
<td>2.29</td>
</tr>
</tbody>
</table>

* In CPU seconds on the University of Texas CDC 6600.
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
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

Anodie, J. and J. Guigou, 1969, Gradient réduit généralisé. Note H1 069/02, Electricité de France.


Van Laar, H.B. and J. Tijms, 1979, Randomly generated polytopes for testing mathematical programming algorithms. Report 7929/0, Econometric Institute, Erasmus University Rotterdam.


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.
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, along with 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 $x_0$, and construct an ellipsoid, $E_0$, 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, $E_k$, of dimension $n-1$. Determine the ellipsoid, $E_{k+1}$, which is tangent to $E_k$ and which has $P'$ as a subset. Then take $x_{k+1}$ as the center of $E_{k+1}$ 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 $2^{3L+33L}$ 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 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, 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.
The process of back-propagation is to compute the gradient of the loss function with respect to the weights of the network. If we denote the loss function as $L(y, \hat{y})$, where $y$ is the true label and $\hat{y}$ is the predicted label, the gradient of $L$ with respect to the weights $w$ can be computed as follows:

$$ \frac{\partial L}{\partial w} = \nabla L \cdot \nabla w $$

Where $\nabla L$ is the gradient of $L$ with respect to the output of the network, and $\nabla w$ is the gradient of the output with respect to the weights. The gradient of the output with respect to the weights can be computed using the chain rule:

$$ \frac{\partial L}{\partial w} = \sum_{i} \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial w} $$

Where $y_i$ is the output of the $i$th neuron. The gradient of the output with respect to the weights can be further decomposed as:

$$ \frac{\partial y_i}{\partial w} = \sum_{j} \frac{\partial y_i}{\partial z_j} \cdot \frac{\partial z_j}{\partial w} $$

Where $z_j$ is the input to the $j$th neuron. The gradient of the output with respect to the weights can be computed as:

$$ \frac{\partial L}{\partial w} = \sum_{i} \sum_{j} \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_j} \cdot \frac{\partial z_j}{\partial w} $$

The back-propagation algorithm iteratively computes these gradients and updates the weights in the opposite direction of the gradient to minimize the loss function.

In the context of neural networks, back-propagation is a key component of training deep learning models. It enables the optimization of the weights in the network by computing the gradient of the loss function with respect to each weight. This gradient information is then used to adjust the weights in a way that reduces the loss function, thereby improving the performance of the model.
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 \( x \) leads to numerical instability very quickly. But, by factoring \( x \) into \( x^2 \), and then applying the transformations directly to \( x \), a numerically stable procedure is obtained. Secondly, the separating hyperplane which, in the Khachiyan method, passes through \( x \), is in 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 \( x^0=0 \); set \( x^0=x \).

If \( Ax < b \), stop; else choose the row \( r \) s.t. \( s=A_rx^k-b \) is greatest.

Transform:

\[
\begin{align*}
a &= x^T \\
e &= s/|J_a| \\
x &= x - (1+e) \cdot J_a^T / (1+n) \cdot J_a \\
J &= \left[I - \frac{1}{1+|n-1|} \cdot \frac{|J_a|}{J_a} \right]
\end{align*}
\]

Go to F.

An algorithm to solve the linear programming problem is:

**Problem:** To find an optimal solution to \( \max cx \), subject to \( Ax=b \), where it is known that if there is an optimal solution at all, there is one such that \( 0 \leq x \).

Phase 1: Find a solution, \( x^0 \), to \( -cx < -y^0 \), \( Ax=b \), \( -x \leq 0 \), (where \( y^0 \) is given by \( -Lx \)); \( x=\{J_c|x \} \), using the ellipsoid algorithm.

Phase 2: Reset \( x^k \) to \( Vx^k \) and find a feasible solution, \( x^k+ \), 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 \( x^*=J_c/|J_c| \). Therefore, if we start with an ellipsoid which contains an optimal solution (so that every succeeding 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

---


---

COMMENTS ON POWELL'S PAPER

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

Besides providing 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 toll, 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.


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 9 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:

\[
\text{max} \sum_{j=1}^{n} 10^{n-j} x_j \quad \text{subject to} \quad x_1 + 2 \sum_{j=1}^{n} 10^{n-j} x_j \leq 10^{2i-2}, i=1,\ldots,n; \quad x_j \geq 0.
\]

Bounds are given by \(x_j \leq 10^{2j-2}\).

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 comparatively 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 53.5 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

\[-15-\]

**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.

REFERENCES

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 [3] 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 behavior of the algorithm when the required solution $x$ 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 standard Fortran. Before discussing the importance of standards, the advantages...
Table 1.

The number of points at which functions and first

derivatives were calculated to solve three test problems.

Table 1.


| Angle | 69 | 69 | 94 |
| Mass | 8 | 8 | 9 |
| Age | 10 | 10 | 10 |

Table 2.

The number of points at which functions and first

derivatives were calculated to solve three test problems.

Table 2.


| Angle | 69 | 69 | 94 |
| Mass | 8 | 8 | 9 |
| Age | 10 | 10 | 10 |

Table 3.

The number of points at which functions and first

derivatives were calculated to solve three test problems.

Table 3.


| Angle | 69 | 69 | 94 |
| Mass | 8 | 8 | 9 |
| Age | 10 | 10 | 10 |
Recent Computational Testing

<table>
<thead>
<tr>
<th>VALUE: 7.75</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLUTION: 1.00 0.75 0.25 0.50 0.25 0.75</td>
</tr>
</tbody>
</table>

Therefore the calculation of $\hat{\Delta}$ is a quadratic programming problem. This remark is an essential part of an algorithm (see [18], for instance), that estimates $B$ automatically, and that uses a line search procedure to force convergence when the initial estimate of $\hat{x}$ 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 [11], however, prefers the name REQP (recursive equality quadratic programming).

Although the solution of a quadratic programming problem on each iteration of a variable metric method for constrained optimization requires much more work than the calculation of the search direction $\hat{\Delta}$ 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 $\hat{\Delta}$ 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 active 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 $\Delta \Delta$ uniquely. Therefore, in contrast 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 $\hat{x}$. Because the "Colville 1" problem 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.
subject to the constraints

\[ \begin{align*}
\lambda \geq 0, \quad \lambda \geq 0, \quad \lambda \geq 0, \\
\end{align*} \]

subject to the constraints

\[ \begin{align*}
\lambda \geq 0, \quad \lambda \geq 0, \quad \lambda \geq 0, \\
\end{align*} \]

subject to the constraints

\[ \begin{align*}
\lambda \geq 0, \quad \lambda \geq 0, \quad \lambda \geq 0, \\
\end{align*} \]
RECENT COMPUTATIONAL TESTING

PROBLEM STATISTICS

<table>
<thead>
<tr>
<th>Method</th>
<th>AVG</th>
<th>SDP</th>
<th>TRNS</th>
<th>TRANS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINS</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>COLS</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>ELEMENTS</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>DENSITY</td>
<td>10</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

TIME RESULTS

<table>
<thead>
<tr>
<th>Iterations</th>
<th>CPU Time (in 3003 sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>0.030 0.110 0.140 0.160</td>
</tr>
<tr>
<td>2nd</td>
<td>0.127 0.200 0.370 0.430</td>
</tr>
<tr>
<td>3rd</td>
<td>0.164 0.233 0.310 0.340</td>
</tr>
<tr>
<td>4th</td>
<td>0.407 0.576 0.820 1.240</td>
</tr>
<tr>
<td>5th</td>
<td>2.734 4.253 57.223</td>
</tr>
</tbody>
</table>

KLEE-DIALCY-CYHATA PROBLEM OF ORDER:

1 5 6 7 8 9 10 11 12 13

ITERATIONS

<table>
<thead>
<tr>
<th>Method</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>6th</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINS</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>COLS</td>
<td>86</td>
<td>119</td>
<td>140</td>
<td>159</td>
<td>165</td>
<td>178</td>
</tr>
<tr>
<td>ELEMENTS</td>
<td>67</td>
<td>77</td>
<td>98</td>
<td>113</td>
<td>131</td>
<td>142</td>
</tr>
<tr>
<td>DENSITY</td>
<td>25</td>
<td>37</td>
<td>49</td>
<td>58</td>
<td>64</td>
<td>66</td>
</tr>
</tbody>
</table>

CPU TIME (in 3003 sec.)

<table>
<thead>
<tr>
<th>Method</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>6th</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINS</td>
<td>0.374</td>
<td>0.407</td>
<td>0.430</td>
<td>0.460</td>
<td>0.497</td>
<td>0.570</td>
</tr>
<tr>
<td>COLS</td>
<td>0.800</td>
<td>1.273</td>
<td>1.493</td>
<td>1.665</td>
<td>1.994</td>
<td>2.357</td>
</tr>
<tr>
<td>ELEMENTS</td>
<td>0.503</td>
<td>0.633</td>
<td>0.747</td>
<td>1.070</td>
<td>1.394</td>
<td>1.700</td>
</tr>
<tr>
<td>LINS</td>
<td>0.606</td>
<td>1.384</td>
<td>8.760</td>
<td>9.496</td>
<td>11.46</td>
<td>15.51</td>
</tr>
<tr>
<td>COLS</td>
<td>4.100</td>
<td>4.903</td>
<td>5.604</td>
<td>7.093</td>
<td>7.768</td>
<td>10.39</td>
</tr>
<tr>
<td>ELEMENTS</td>
<td>5.100</td>
<td>6.333</td>
<td>7.000</td>
<td>8.153</td>
<td>9.200</td>
<td>11.93</td>
</tr>
<tr>
<td>DENSITY</td>
<td>1.600</td>
<td>3.333</td>
<td>1.513</td>
<td>3.200</td>
<td>6.54</td>
<td>18.62</td>
</tr>
<tr>
<td>LINS</td>
<td>0.767</td>
<td>1.777</td>
<td>3.593</td>
<td>5.196</td>
<td>9.526</td>
<td>16.56</td>
</tr>
<tr>
<td>COLS</td>
<td>1.003</td>
<td>2.277</td>
<td>4.210</td>
<td>7.437</td>
<td>11.65</td>
<td>19.08</td>
</tr>
</tbody>
</table>

FEATURE ARTICLE

The method employs an algorithm for unconstrained optimization to calculate the vector \( x \) that minimizes the function

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

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

\[
F(x) - \sum_{i=1}^{m} \lambda_i^* c_i(x) \quad (3.3)
\]

is stationary at the required solution, \( x^* \) say. Therefore, for all values of \( r \), the function (3.2) is also stationary at \( x = x^* \) if \( \lambda = \lambda^* \). 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 \( x(\lambda^*, r) \) equal to \( x^* \). Then \( r \) is fixed, and, by using the calculated vectors \( x(\lambda, r) \), the parameters \( \lambda \) are adjusted automatically so that they converge to \( \lambda^* \). Hence \( x(\lambda, r) \) converges to the required solution. Three good properties of the method are...
The text is not legible due to the quality of the image.
3. The problem file. Nine typical lines of the problem file are as follows:

*ROSENBEK THIS IS THE EVER-FAMOUS ROSENBECK FUNCTION.
0001 0001 0002 00100
-1.2 1.0

*ROSENBEK2 THIS IS ROSENBECK'S FUNCTION WITH A DIFFERENT XO.
0002 0001 0002 00100
2.5 -1.6

HELIIX 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 ROSENBEK. 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 ROSENBEK2 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 (FUNCTION, N, X, F, G, IFL): This calls FUNCTION(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 \| \leq EPS, or setting VEC1 = \| x \| and VEC2 = \| x_{k+1} \|, one may test if \| x_k - x_{k+1} \| \leq EPS. One may choose p = 1,2 or \infty. 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.

Because of the delays that have just been mentioned, a review of published 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 structure. Among the other algorithms for this calculation, there are two that were proposed by Davidson 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 matrix.

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 strong numerical evidence that suggests that either algorithm is better than the BFGS algorithm in practice, but little experience has been obtained so far with Davidson'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
-52-

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:

THE FUNCTION VALUE IS .104000000E+03 (1 FUNCTION EVALUATIONS).
THE GRADIENT AT THIS POINT IS .200000000E+03.

THE VARIABLES HAVE THE CURRENT VALUES GIVEN BY
-1.0000000E+01 0.
THE GRADIENT AT THIS POINT IS
-3.0400000E+03.

THE SOLUTION HAS BEEN FOUND:

THE FUNCTION VALUE IS .283000000E-15 (38 FUNCTION EVALUATIONS).
THE VARIABLES HAVE THE CURRENT VALUES GIVEN BY
1.0000000E+00 0.
THE GRADIENT AT THIS POINT IS
2.7976000E-08.

SUMMARY OF PROBLEM:
PR # NAME DIM ITS FNC GRDS PVALUE GVALUE MSECS FSECS
1 13 HELIX 3 27 38 .327E-15 .30E-06 .17 .01 0
1 1 ROSENBRK 2 38 44 .378E-13 .68E-05 .19 .01 0
1 5 OREN20 20 100 100 .19E-02 .70E-01 5.19 .13 1

SUMMARY OF PROBLEMS DONE:
PR # FN NAME DIM ITS FNC GRDS PVALUE GVALUE MSECS FSECS
1 13 HELIX 3 27 38 .327E-15 .30E-06 .17 .01 0
1 1 ROSENBRK 2 38 44 .378E-13 .68E-05 .19 .01 0
1 5 OREN20 20 100 100 .19E-02 .70E-01 5.19 .13 1

THERE WERE 1 PROBLEMS FLAGGED WITH ERRORS.

FEATURE ARTICLE

Problems 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 method 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
To further generalize this concept, consider a model that has been trained on a dataset containing images of various objects. Suppose we have a model that can classify images into different categories, such as animals, vehicles, and furniture. This model has been trained using a large dataset of images, and it has learned to recognize patterns in the data that are characteristic of each category.

When we test this model on new images, we can see that it performs well on images that are similar to those it was trained on. However, if we present the model with images of objects that are quite different from what it has seen before, it may struggle to make accurate predictions. This is where transfer learning comes into play. By fine-tuning the model on a new dataset, we can adapt it to recognize new categories of objects.

For example, let's say we want to train our model to identify birds. We can use transfer learning to fine-tune the model on a dataset of bird images. The model will then be able to make accurate predictions about bird images, even if it has never seen one before.

This is why transfer learning is so powerful. It allows us to leverage the knowledge that we have already learned from previous tasks and apply it to new, unseen problems. By doing so, we can create more robust and versatile models that are better able to generalize to new data.

In conclusion, transfer learning is a valuable technique that can help us create more effective and efficient machine learning models. By reusing the knowledge that we have already learned, we can build models that are better able to generalize to new data, which is crucial for many real-world applications.
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.


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 \( F(x) \) 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.
This article is reprinted from Lecture Notes in Control and Information Sciences.

The algorithms that do not require derivations are calculated.

Vorsatz can be calculated for any X. If we suppose that all functions are real, then their total on the values of the vertices, we reserve n for the number of components.

\[ f(X, \bar{X}) = \sum_{i=1}^{n} g_i(X_i, \bar{X}_i) \]

\[ g_i(X_i, \bar{X}_i) = \begin{cases} 1 & X_i = 1 \\ 0 & \text{otherwise} \end{cases} \]

\[ f(X, \bar{X}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} X_i > n/2 \\ 0 & \text{otherwise} \end{cases} \]
1982

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

August 23-28: Eleventh International Symposium on Mathematical Programming in Bonn, Federal Republic of Germany. Contact: Institut für Ökonomie und Operations Research Universität Bonn. Nussallee 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 Prinzes 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 Halvey, School of Engineering/Applied Science, Princeton University, Princeton, New Jersey 08540.

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
This newsletter is mailed to every member of the Professional Engineering Community of the University of Kansas. If you are not a member, we encourage you to join, as a member of the community, a forum for the exchange of ideas and experiences.

NEWSLETTER CONTENTS

1. NEWSLETTER ORIGINS

2. NEWSLETTER CONTENTS

3. NEWSLETTER ORIGINS

This newsletter is mailed to every member of the Professional Engineering Community of the University of Kansas. If you are not a member, we encourage you to join, as a member of the community, a forum for the exchange of ideas and experiences.
Mathematical Programming Society
Committee on Algorithms Newsletter

No. 5: February 1981

Karla L. Hoffman, Editor

Contents:

Editor's Column - Karla L. Hoffman

Feature Article
Optimization Algorithms in 1979
M. J. D. Powell

Comments on Powell's Article
John N. Mulvey

Richard H. F. Jackson

Recent Computational Testing
An Evaluation and Comparison of Curve Fitting Solutions
Rosemary Chang

An Evaluation of Max Flow Algorithms
Fred Glover, Darren Kilman, and John Nocle

Computational Methods for Minimum Spanning Tree Methods
R. E. Raymond, J. P. Jarvis, and D. R. Skier

Implementing a Mathematical Programming Algorithm: Performance
Considerations and A Case Study
Uwe Suhl

A Computational Comparison of 2-Phase Algorithms and a Recursive
Quadratic Programming Algorithm
G. van der Hook

Computational Experience with Ellipsoidal Algorithms for Linear
Programming
A. C. Williams

Other COAL-Related Research Activities
Testing Minimization Codes
A. Buckley

A Note on a Small Model Which Cycled
C. A. Haverly

Calendar of Mathematical Programming Meetings

1
2
17
18
20
21
26
27
28
37
49
53
55