Mathematical Programming 10 (1976) 52–69.
North-Holland Publishing Company

GLOBAL OPTIMIZATION USING SPECIAL ORDERED SETS

E.M.L. BEALE and J.J.H. FORREST
Scientific Control Systems Ltd., Milton Keynes, Great Britain

Received 26 November 1973
Revised manuscript received 20 May 1975

The task of finding global optima to general classes of nonconvex optimization problem is attracting increasing attention. McCormick [4] points out that many such problems can conveniently be expressed in separable form, when they can be tackled by the special methods of Falk and Soland [2] or Soland [6], or by Special Ordered Sets. Special Ordered Sets, introduced by Beale and Tomlin [1], have lived up to their early promise of being useful for a wide range of practical problems. Forrest, Hirst and Tomlin [3] show how they have benefitted from the vast improvements in branch and bound integer programming capabilities over the last few years, as a result of being incorporated in a general mathematical programming system.

Nevertheless, Special Ordered Sets in their original form require that any continuous functions arising in the problem be approximated by piecewise linear functions at the start of the analysis. The motivation for the new work described in this paper is the relaxation of this requirement by allowing automatic interpolation of additional relevant points in the course of the analysis.

This is similar to an interpolation scheme as used in separable programming, but its incorporation in a branch and bound method for global optimization is not entirely straightforward. Two by-products of the work are of interest. One is an improved branching strategy for general special-ordered-set problems. The other is a method for finding a global minimum of a function of a scalar variable in a finite interval, assuming that one can calculate function values and first derivatives, and also bounds on the second derivatives within any subinterval.

The paper describes these methods, their implementation in the UMPIRE system, and preliminary computational experience.

1. Introduction

The task of finding global optima to general classes of nonconvex optimization problems is attracting increasing attention. McCormick [4] discusses various approaches, and points out that many of these problems can conveniently be expressed in separable form, when they can

be tackled by the special methods of Falk and Soland [2] or Soland [6] or by Special Ordered Sets.

Special Ordered Sets were introduced by Beale and Tomlin [1]. They have lived up to their early promise of being useful for a wide range of practical problems. They are of two types. Special ordered sets of Type 1, or S1 sets, are sets of variables of which not more than one member may be nonzero in the final solution. Special ordered sets of Type 2, or S2 sets, are sets of variables of which not more than two members may be nonzero in the final solution, with the further condition that if there are as many as two they must be adjacent.

S2 sets were introduced to make it easier to find global optimum solutions to problems containing piecewise linear approximations to nonlinear functions of single arguments in otherwise linear programming problems. We may want to introduce some function $a(t)$ into either a constraint or the objective function, where the argument t is some linear function of other variables of the problem. If $a(t)$ is approximately piecewise linear between the points $t = T_0, T_1 \ldots T_J$, then we proceed as follows: Define a set of nonnegative variables λ_j for $j = 0, \ldots, J$ and the constraints

$$\sum_j \lambda_j = 1, \tag{1.1}$$

$$\sum_j T_j \lambda_j - t = 0, \tag{1.2}$$

and then the nonlinear function $a(t)$ is represented by the linear function

$$\sum_j a(T_j) \cdot \lambda_j, \tag{1.3}$$

provided that not more than two of the λ_j are allowed to be nonzero, and if there are as many as two they are adjacent.

In practice, problems of this kind usually involve several functions of different arguments, which we may denote by t_k. And for any argument t_k we may have more than one nonlinear function. So we may write $a_{ik}(t_k)$ as the function of the k^{th} argument occurring in the i^{th} row of the problem. If all $a_{ik}(t_k)$ are approximately piecewise linear functions of t_k between the points $t_k = T_{0k}, T_{1k}, \ldots, T_{J_k k}$, then we

proceed as follows:

Define k sets of nonnegative variables λ_{jk} for $j = 0, \ldots, J_k$ and all k, and the constraints

$$\sum_j \lambda_{jk} = 1 \quad \text{for all } k\,, \tag{1.4}$$

$$\sum_j T_{jk}\lambda_{jk} - t_k = 0 \quad \text{for all } k\,, \tag{1.5}$$

and then each nonlinear function $a_{ik}(t_k)$ is represented by the linear function

$$\sum_j a_{ik}(T_{jk}) \cdot \lambda_{jk}\,, \tag{1.6}$$

provided that, for each k, not more than two of the λ_{jk} are allowed to be nonzero and if there are as many as two they are adjacent.

This approach was first introduced by Miller [5] as the so-called λ-formulation of separable programming. Separable programming finds local optimum solutions to such problems. Various methods have been proposed for adding further constraints and integer variables, and hence finding global optimum solutions. But it is simpler and more efficient to operate directly on the λ-variables as an S2 set.

Forrest, Hirst and Tomlin [3] show how special ordered sets have benefitted from the vast improvements that have recently been made in general integer programming capabilities using branch and bound methods. This has happened because special ordered sets have been incorporated in a general mathematical programming system, rather than because explicit efforts have been made to relate new concepts, such as pseudo-costs and degradations, to special ordered sets. It now seems appropriate to reconsider all these things, particularly as special ordered sets originally required that any continuous functions arising in the problem had to be approximated by piecewise linear functions in the initial input to the mathematical programming system.

The problem of finding global optima to problems involving general functions of single arguments with continuous derivatives is straightforward in principle: one uses S2 sets, and interpolates where necessary to improve the accuracy to which each nonlinear function is approximated. There remain the questions when and how to branch on S2 sets,

and when and how to branch on any S1 sets or integer variables that may be present in the problem. This paper suggests answers to all these questions. After defining some notation and terminology in Section 2, we consider both when and where to branch on special ordered sets or integer variables in Section 3. This material applies whether or not the problem requires interpolation. Then in Section 4 we consider interpolation. Finally, preliminary computational experience is discussed in Section 5.

The Appendix is devoted to the problem of finding a global minimum of a function of one variable in a bounded interval. This is of some interest in its own right, and is vital to the success of the interpolation phase. We describe a solution based on calculating function values and first derivatives, and also bounds on the second derivatives in any subinterval. This has proved satisfactory for our present purposes.

2. Notation and terminology

The notation and terminology are based on ordinary mathematical programming usage, but some special notation and terminology are convenient.

We use the subscript i to refer to a row, and the subscript k to refer to a Special Ordered Set.

The rows (1.4) and (1.5) are known respectively as the convexity row and the reference row for the k^{th} Special Ordered Set.

It seems most natural to define the columns within an S2 set as functions of the reference row entry t_k, rather than identifying them by subscripts that cannot easily be given meaningful numerical values. So the variables in the k^{th} set are denoted by $x_k(t_k)$. This corresponds to the variable denoted by λ_{jk} in Section 1 if $t_k = T_{jk}$. The optimum values of the $x_k(t_k)$ in any linear programming subproblem are denoted by $x_k^{\circ}(t_k)$.

The vector of coefficients of the variable $x_k(t_k)$ is denoted by $\boldsymbol{a}_k(t_k)$, and its i^{th} component, representing the coefficient in the i^{th} row, is denoted by $a_{ik}(t_k)$.

We let $i_{\mathrm{R}k}$ denote the reference row for the k^{th} set. Since the coefficient of $x_k(t_k)$ in this row is t_k, we have the equation

$$a_{i_{\mathrm{R}k}k}(t_k) = t_k \ .$$

We use the symbol Σ_k to denote summation over all variables in the k^{th} set. In this notation the convexity row normally takes the form

$$\sum_k x_k(t_k) = 1 ,$$

as indicated by (1.4), but some formulations require the more general form

$$\sum_k x_k(t_k) + s_{Sk} = b_{Sk} ,$$

where s_{Sk} is a nonnegative slack variable. We therefore develop the theory for this more general type of convexity row, although we will in practice usually have $b_{Sk} = 1$ and the slack on the convexity row will usually not exist, in which case the optimum value s°_{Sk} in any linear programming subproblem will be zero.

We now define the "average reference row entry" $\bar{t}_k$ for the k^{th} Special Ordered Set. This can be interpreted as the value of the k^{th} argument implied by the solution to the linear programming subproblem. A formula that applies in all cases is

$$\bar{t}_k = \sum_k t_k x^{\circ}_k(t_k) / \sum_k x^k_{\circ}(t_k)$$

$$= \sum_k t_k x^{\circ}_k(t_k) / (b_{Sk} - s^{\circ}_{Sk}) .$$

We may also define the "end points of the current interval", t^-_k and t^+_k as the values of t_k for which $x_k(t_k)$ are the consecutive members of the set such that

$$t^-_k \leqslant \bar{t}_k \leqslant t^+_k .$$

In practice this is only important for S1 sets, since if an S2 set contains only a finite number of members we can conceptually add further members by linear interpolation between adjacent members.

Now corresponding to any linear programming subproblem we can define three approaches to the contribution of the variables in the k^{th}

set to the problem. There is the vector a_{Ak} of actual contributions defined by

$$a_{Ak} = -\sum_k x_k^{\circ}(t_k) \cdot a_k(t_k) .$$

For an S2 set, there is the vector a_{Ck} of corrected contributions corresponding to the average reference row entry and the value of the slack on the convexity row defined by

$$a_{Ck} = (b_{Sk} - s_{Sk}^{\circ})\, a_k(\bar{t}_k) .$$

For an S1 set we may define corrected contributions a_{Ck}^{-} and a_{Ck}^{+} corresponding to the two end points of the current interval by the formulae

$$a_{Ck}^{-} = (b_{Sk} - s_{Sk}^{\circ})\, a_k(t_k^{-})$$

and

$$a_{Ck}^{+} = (b_{Sk} - s_{Sk}^{\circ})\, a_k(t_k^{+}) .$$

A third approach is relevant when we have to decide where to branch in the k^{th} set. If t_{kS} and t_{kE} denote the smallest and largest values of t_k for which $x_k^{\circ}(t_k) > 0$, then we define the vector $a_{Ik}(t_k)$ of interpolated coefficients corresponding to any t_k between t_{kS} and t_{kE} by

$$a_{Ik}(t_k) = (1-\theta)\, a_k(t_{kS}) + \theta\, a_k(t_{kE}) ,$$

where θ is defined by the equation

$$t_k = (1-\theta)\, t_{kS} + \theta\, t_{kE} .$$

The discrepancy between $a_{Ik}(t_k)$ and $a_k(t_k)$ can be regarded as an indication of the extent to which the current linear programming subproblem misrepresents the consequences of giving the k^{th} argument the value t_k.

As usual we let π_i denote the shadow price on the i^{th} row. This refers to the ultimate objective function if the linear programming subproblem is feasible, and to the sum of infeasibilities otherwise. We denote

the reduced cost of the variables $x_k(t_k)$ by $d_k(t_k)$, so

$$d_k(t_k) = \sum_i \pi_i a_{ik}(t_k) .$$

The term "degradation" is used as elsewhere in the integer programming literature to mean the reduction in the value of the objective function in a subproblem caused by imposing the additional condition that a specified integer variable or special ordered set must be satisfied, i.e., that the integer variable must take an integer value or that the members of the special ordered set take feasible values.

3. Branching on special ordered sets and integer variables

The logic originally proposed by Beale and Tomlin for branching on S2 sets is unsuitable for S2 sets containing an arbitrarily large number of elements representing functions with continuous derivatives. And it now seems that this logic could usefully be changed for all special ordered sets. The objection to the original logic is that it relies on the average reference row entry $\bar{t}_k$ remaining near the value obtained at the source node of the branch, and correspondingly pays too little attention to minimizing the discrepancies between the linearly interpolated and true values of the entries in each row over the range of possible values of t_k.

We therefore re-examine the problem, which we can divide into 3 parts:

(a) How can we estimate the degradation that will result from replacing the current variables from a Special Ordered Set by a valid variable or pair of variables?

(b) How can we decide which Special Ordered Set to branch on, or whether we should instead branch on an integer variable?

(c) Having decided to branch on a particular Special Ordered Set, how do we decide where the branch should be made?

To answer part (a) for an S2 set, we examine the vector $a_{Ck} - a_{Ak}$. A crude estimate of the degradation resulting from using the corrected contributions instead of the actual contributions is then

$$\sum_i \pi_i (a_{Cik} - a_{Aik}) .$$

But this quantity may underestimate the degradation, since the shadow

prices only measure the costs of infinitessimal changes in the contributions to the corresponding rows, and may therefore underestimate the costs of finite changes in these contributions. In particular, the degradation estimated from this formula could be zero in situations where further branching is necessary.

To overcome this difficulty, two further sets of nonnegative input quantities p_i^+ and p_i^- should be defined, representing pseudo shadow prices. These are related to pseudo costs, which have proved useful in integer programming. They should be such that $p_i^+ = 0$ whenever the i^{th} row has a negative slack or is an objective function, and $p_i^- = 0$ whenever the i^{th} row has a positive slack or is an objective function. Otherwise p_i^+ and p_i^- should be strictly positive. Good estimates of these quantities should be used when available, but underestimates are less disastrous than overestimates so we have set the default value to 10^{-4}.

The approximate degradation D_{Ak} that will be incurred removing the infeasibility in the k^{th} set, assuming it is an S2 set, is then defined by

$$D_{Ak} = \sum_i \max\{p_i^+(a_{Cik} - a_{Aik}), -p_i^-(a_{Cik} - a_{Aik}), \pi_i(a_{Cik} - a_{Aik})\}.$$

Note that this formula can be written as

$$D_{Ak} = \sum_i C_i ,$$

where

$$C_i = \max(p_i^+, \pi_i) \cdot (a_{Cik} - a_{Aik}) \quad \text{if } a_{Cik} - a_{Aik} \geqslant 0 ,$$
$$= \max(p_i^-, -\pi_i)\,|a_{Cik} - a_{Aik}| \quad \text{if } a_{Cik} - a_{Aik} < 0 .$$

Similarly the approximate degradation D_{Ak} that will be incurred removing the infeasibility in the k^{th} set if it is an S1 set can be defined as

$$D_{Ak} = \min(D_{Ak}^-, D_{Ak}^+) ,$$

where

$$D_{Ak}^- = \sum_i \max\{p_i^+(a_{Cik}^- - a_{Aik}), -p_i^-(a_{Cik}^- - a_{Aik}), \pi_i(a_{Cik}^- - a_{Aik})\}.$$

(note that these formulae include the pseudo shadow prices on the reference row) and

$$D^{+}_{\mathrm{A}k} = \sum_i \max\{p^{+}_i (a^{+}_{\mathrm{C}ik} - a_{\mathrm{A}ik}), -p^{-}_i (a^{+}_{\mathrm{C}ik} - a_{\mathrm{A}ik}), \pi_i (a^{+}_{\mathrm{C}ik} - a_{\mathrm{A}ik})\}.$$

We can estimate the degradation incurred in giving an integer variable an integer value in the same way if we imagine that the problem has been reformulated with an S1 set having a member for each possible value of the integer variable. Specifically, if x_j is an integer variable, we could define an S1 set such that

$$a_{ik}(t_k) = a_{ij} \cdot t_k \ ,$$

where t_k can only take integer values between the lower and upper bounds on x_j, and where $b_{\mathrm{S}k} = 1$ and $s_{\mathrm{S}k} \equiv 0$.

In this case there will not be a reference row in the normal sense, but it is convenient to treat the simple upper bound row restricting the range of possible values of x_j as a reference row. The actual shadow price on this row will be zero if the integer variable is not satisfied, and the pseudo shadow price on this row play roles similar to those of pseudo costs in other approaches to integer programming. But these pseudo shadow prices are not the same as pseudo costs, since the estimated degradation will be affected by the pseudo shadow prices and actual shadow prices on other rows in which the integer variable has coefficients.

To answer part (b) of our original question, we follow the rule of branching on the integer variable or special ordered set with the greatest estimated degradation.

To answer part (c), we compare the coefficients $a_{ik}(t_k)$ with the interpolated coefficients $a_{\mathrm{I}ik}(t_k)$. We define the function $D_{\mathrm{I}k}(t_k)$ by

$$D_{\mathrm{I}k}(t_k) = \sum_i \max\{p^{+}_i (a_{ik}(t_k) - a_{\mathrm{I}ik}(t_k)), -p^{-}_i (a_{ik}(t_k) - a_{\mathrm{I}ik}(t_k)), \pi_i (a_{ik}(t_k) - a_{\mathrm{I}ik}(t_k))\}.$$

When branching on the k^{th} set, it is natural to branch at a value t^{M}_k of t_k that maximizes $D_{\mathrm{I}k}(t_k)$. If this function has more than one local

maximum, we can accept any for which $D_{Ik}(t_k) > 0$. If the set is an S1 set, we branch in such a way as to exclude an interval with one end point at t_k^M and the other end point on the same side of t_k^M as $\bar{t}_k$. Some encouraging mathematical properties of this procedure can easily be established:

(a) If $D_{Ak} = 0$, then the k^{th} set is satisfied, although some slack variables may be underestimated and some objective functions other than the current one may be misestimated.

(b) If the functions $a_{ik}(t_k)$ are all linear over some interval, then the criterion can never choose to branch at a point t_k^M within such an interval, rather than at an end point of the interval.

(c) For S2 sets, in the common situation where $a_{ik}(t_k)$ is zero or a linear function of t_k for all i but one, and is convex or concave for this one i, then the recommended choice of branching point minimizes the average value of $D_{Ik}(t_k)$ within each of the two new subproblems created, where the average is taken over all values of t_k permitted in the current subproblem.

4. Interpolation

In any branch and bound process, it is useful to have a good bound on the value of the objective function in any subproblem. As in Dantzig–Wolfe decomposition, we know that if we solve a linear programming subproblem omitting some variables from S2 sets, then a valid upper bound on the solution is given by the current value of the linear programming optimum plus

$$\sum_k \max_{t_k}(-d_k(t_k)) \cdot b_{Sk} .$$

We may be able to reduce this bound by trying to attain it. So we introduce that member $x_k(t_k)$ of the set giving the most negative reduced cost $d_k(t_k)$, provided that the corresponding $d_k(t_k)b_{Sk}$ is less than some threshold value. This part of the process requires a global minimization which is not necessarily trivial: but a suitable method is developed in the Appendix.

One variable from each S2 set is introduced in this way, provided that the most negative reduced cost is sufficiently negative, whenever

the subproblem has been optimized using the currently available variables. This process continues until either

(a) the bound falls below the value of the objective function at the best trial solution so far, in which case the subproblem can be abandoned,

(b) $d_k(t_k) \cdot b_{Sk}$ is greater than or equal to the threshold value for all variables in all sets.

5. Results

Table 1 illustrates the effect of the new methods on two problems containing S2 sets not requiring interpolation. Problem A is a small problem where the sets include discontinuities. Problem B is a larger problem. Both illustrate the fact that the new branching strategy substantially reduces the number of branches needed for a complete search and that the increased overhead in computation is minimal.

Table 2 illustrates the use of interpolation on Problem B. All three runs used the new branching strategy, but the first had enough vectors in each S2 set defined in the initial matrix to ensure the required accuracy. The other two runs started with only two vectors in each S2 sets at the end points of the permitted range and used automatic interpolation to create others as required. The second run was required to to achieve the same accuracy in the representation of the nonlinear functions as the first. The third run required 1000 times more accuracy. It is interesting that even when there are relatively few vectors in each set, the ability to branch at the very best point according to our criteria

Table 1
Computational results for problems including S2 sets not requiring interpolation.

Problem	A		B	
Rows	65		881	
Sets	24		17	
Vectors in sets	144		431	
Other vectors	42		1570	
Strategy	Old	New	Old	New
Branches	50	22	298	122
Time [a] for Complete Search	37 sec	18 sec	90 min	37 min

[a] The times are CPU times on a Univac 1108.

Table 2
Computational results for a problem including S2 sets with a fixed grid and with automatic interpolation.
Problem B 881 rows, 17 sets, 1570 vectors outside sets.

Method of approximation	Piecewise linear approximation	Function with automatic interpolation	Function with automatic interpolation
Maximum error	1.0	1.0	0.001
Vectors needed to achieve required accuracy over complete ranges	431	431	11038
Vector (initially)	431	34	34
Vectors (at optimum)	431	78	106
Branches	122	92	192
Time [a] (minutes)	37	28	65

[a] The times are CPU times on a Univac 1108.

made the second run much quicker than the first. If increased accuracy is needed, the fixed grid approach would be excessively expensive since it would need over 10000 vectors in the S2 sets alone.

Appendix. Global optimization in one dimension

The problem of minimizing a function $f(x)$ of a scalar argument x within a finite interval $L \leqslant x \leqslant U$ has been studied extensively, particularly during the last 15 years. Methods such as golden section search have been developed for finding a minimum to arbitrarily high accuracy. But these methods are only applicable if we are content with a local minimum, or if the function is convex.

Our problem is to find a value of x that gives $f(x)$ a value within some tolerance ϵ of its global minimum when it is not necessarily convex. For arbitrary functions there is no finite method for this. But we can derive a finite method under some additional assumptions that cover our applications. If we use a prime to denote differentiation with respect to x, we can write these assumptions as

(a) The function $f(x)$ is twice differentiable,

(b) we can evaluate f and f' for any x within $L \leqslant x \leqslant U$,

(c) we can derive finite upper and lower bounds on f'' within any interval.

To this extent our assumptions, and our methods, follow those pioneered by Richard Brent. But these methods seem of very limited value unless we have a convenient method of finding bounds on f''. So we discuss this point first.
We assume that

$$f(x) = \sum_i \pi_i a_i(x) , \tag{A1}$$

and we assume that the total interval $L \leqslant x \leqslant U$ is divided into a finite number of subintervals such that each $a_i''(x)$ is monotonic within each subinterval. This assumption follows naturally from the fact that the $a_i(x)$ must be calculable within a general mathematical programming system. Most of the functions with which we are concerned have second derivatives that change monotonically over the entire range of values of x for which the function is continuous. Others, such as polynomials of degree higher than three, can conveniently be represented as sums of functions with monotonic second derivatives.

We now have the inequalities that, for $l_j \leqslant x \leqslant u$, $M_1 \leqslant f'' \leqslant M_2$, where

$$M_1 = \sum_i \min(\pi_i a_i''(l_j), \pi_i a_i''(u)) ,$$

and

$$M_2 = \sum_i \max(\pi_i a_i''(l_j), \pi_i a_i''(u)) . \tag{A2}$$

So much for the assumptions. We now give a geometrical outline of our algorithm. We start by calculating f at the end points of all subintervals over which the a_i'' are monotonic and define $f_{\min}$ as the smallest of these values of f and $x_{\min}$ as the corresponding value of x.

We must now see whether there is any possibility that $f < f_{\min} - \epsilon$ within any subinterval. We therefore compute M_1 and M_2 for the subinterval from (A2) and construct a parabola with second derivative defined by M_1 and with the same function value and first derivative as f at the lower end of the subinterval. This parabola defines a lower bound on the value of $f(x)$ anywhere within the subinterval. We find another lower bound by constructing another parabola with the same function value and first derivative as f at the upper end of the subinterval. We now sharpen the bounds by constructing another parabola with second

derivative M_2 touching each of the first two parabolae. The piecewise parabolic function $g(x)$ defined by these three parabolae then defines a lower bound for f for any x within the subinterval. If the minimum of this lower bound function exceeds $f_{\min} - \epsilon$, then we can exclude this subinterval and proceed to the next one. If not, we divide the subinterval into two further subintervals at the point where the lower bound function is minimized.

Before proceeding to the algebraic details of the algorithm it is worth discussing two practical points that make the logic a little more complicated but which speed up the process in practice. We note that if $M_2 \leqslant 0$ within a subinterval, then f is concave within this subinterval and there is no possibility of an interior minimum. Equally, if $M_1 \geqslant 0$, then f is convex and we need only search for a local minimum within this subinterval.

We now derive formulae for the minimum of the function $g(x)$ for $0 \leqslant x \leqslant h$ under the conditions that

$$g(0) = f(0), \qquad g'(0) = f'(0),$$
$$g(h) = f(h), \qquad g'(h) = f'(h),$$
$$M_1 \leqslant g'' \leqslant M_2 .$$

It is fairly obvious that the minimum is achieved by putting $g'' = M_1$ for $0 < x < x_1$ and $x_2 < x < h$, and $g'' = M_2$ for $x_1 < x < x_2$, for some x_1 and x_2. We therefore derive formulae for x_1 and x_2 based on continuity requirements. We see that

$$g'(x_1) = f'(0) + M_1 x_1$$

and

$$g(x_1) = f(0) + f'(0) \cdot x_1 + \tfrac{1}{2} M_1 x_1^2 .$$

Hence

$$g'(x_2) = g'(x_1) + M_2(x_2 - x_1)$$
$$= f'(0) + (M_1 - M_2)x_2 + M_2 x_2 ,$$

and

$$g(x_2) = g(x_1) + g'(x_1) \cdot (x_2 - x_1) + \tfrac{1}{2} M_2 (x_2 - x_1)^2 ,$$

which reduces to

$$g(x_2) = f(0) + f'(0) \cdot x_2 + \tfrac{1}{2}(M_2 - M_1)x_1^2$$
$$+ (M_1 - M_2)x_1 x_2 + \tfrac{1}{2} M_2 x_2^2 .$$

Hence

$$g'(h) = g'(x_2) + M_1(h - x_2)$$
$$= f'(0) + (M_1 - M_2)(x_1 - x_2) + M_1 k\,, \qquad \text{(A3)}$$

and

$$g(h) = g(x_2) + g'(x_2) \cdot (h - x_2) + \tfrac{1}{2} M_1 (h - x_2)^2\,,$$

which reduces to

$$g(h) = f(0) + f'(0) \cdot h + \tfrac{1}{2} M_1 h^2$$
$$+ \tfrac{1}{2}(M_1 - M_2)(x_2 - x_1)(x_2 + x_1 - 2h)\,. \qquad \text{(A4)}$$

But $g'(h) = f'(h)$ and $g(h) = f(h)$, so if we write

$$a = f'(h) - f'(0) - M_1 h\,,$$
$$b = f(h) - f(0) - f'(0) \cdot h - \tfrac{1}{2} M_1 h^2\,,$$

we deduce from (A3) that

$$x_2 - x_1 = a/(M_2 - M_1) \qquad \text{(A5)}$$

and from (A4) that

$$b = \tfrac{1}{2}(M_1 - M_2)(x_2 - x_1)(x_2 + x_1 - 2h)$$
$$= \tfrac{1}{2} a\,(2h - x_1 - x_2)\,.$$

so that

$$x_1 + x_2 = 2h - 2b/a\,. \qquad \text{(A6)}$$

Hence, from (A5) and (A6),

$$x_1 = h - b/a - \tfrac{1}{2} a(M_2 - M_1)\,.$$

We also write

$$c = f'(0) + M_1 x_1\,.$$

Then, for $x_1 < x < x_2$,

$$g'(x) = c + M_2(x - x_1),$$

so $g'(x) = 0$ when $x = x_1 - c/M_2$, and at this point

$$g(x) = f(0) + f'(0)\,x_1 + \tfrac{1}{2}M_1 x_1^2 + c(x - x_1) + \tfrac{1}{2}M_2(x - x_1)^2,$$

which reduces to

$$g(x) = f(0) + \tfrac{1}{2}f'(0)\,x_1 + \tfrac{1}{2}c(x_1 - c/M_2). \qquad \text{(A7)}$$

The right-hand side of (A7) defines the minimum of $g(x)$ if

$$0 \leqslant -c/M_2 \leqslant x_2 - x_1 = a/(M_2 - M_1).$$

If $-c/M_2$ lies outside this range, then no interior minimum is possible unless $M_1 > 0$. But if $M_1 > 0$ and $c > 0$ the minimum is

$$f(0) - \tfrac{1}{2}\{f'(0)\}^2/M_1, \text{ taken when } x = -f'(0)/M_1,$$

while if $M_1 > 0$ and $-c/M_2 > a/(M_2 - M_1)$ the minimum is

$$f(h) - \tfrac{1}{2}\{f'(h)\}^2/M_1, \text{ taken when } x = h - f'(h)/M_1.$$

We can now define the algorithm for the global minimization of the function $f(x)$ of the scalar variable x within the range $L \leqslant x \leqslant U$. We assume that $f(x)$ is defined by (A1) and that each $a_i''(x)$ is monotonic within any subinterval $p_k < x < p_{k+1}$ for any $k < K$, where

$$L = p_1 < p_2 \ldots < p_K = U.$$

We also assume that we are given a positive tolerance ϵ.

We start by defining $l_k = p_k$ for $k = 1, \ldots, K-1$; and compute and store $f(x)$, $f'(x)$ and $a_i''(x)$ for $x = l_k$. We put $u = U$ and compute and

store $f(u)$, $f'(u)$ and $a_i''(u)$. Put

$$f_{\min} = \min(\min f(l_k), f(u))$$

and define

$$x_{\min} \text{ such that } f(x_{\min}) = f_{\min} .$$

The algorithm also uses an indicator I_k, which is set to 1 if $f(x)$ is convex in an interval spanning l_k and which is set to 0 otherwise. We put $I_1 = 1, I_k = 0$ for $1 < k \leqslant K-1$.

Set $j = K-1$ and enter a general step of the algorithm, which is as follows: Compute

$$M_1 = \sum_i \min(\pi_i a_i''(l_j), \pi_i a_i''(u))$$

$$M_2 = \sum_i \max(\pi_i a_i''(l_j), \pi_i a_i''(u)) .$$

If $M_2 \leqslant 0$ go to (*).

(*) Come here if no appreciably better solution can exist for $l_j < x < u$. We then put $u = l_j$. If $j = 1$ the whole problem is solved and $f_{\min} = f(x_{\min})$ is within ϵ of the minimum value of $f(x)$. Otherwise put $j = j-1$ and start a new step of the algorithm.

Otherwise proceed as follows: Compute

$$h = u - l_j ,$$
$$t = f'(l_j) + M_1 h ,$$
$$a = f'(u) - t ,$$
$$b = f(u) - f(l_j) - \tfrac{1}{2} h (f'(l_j) + t) .$$

If $ah < \epsilon$, then if $M_1 \leqslant 0$ or if $f'(l_j) \geqslant 0$, or $t \leqslant 0$, go to (*). Otherwise put

$$x_T = -f'(l_j)/M_1 ,$$
$$f_T = f(l_j) + \tfrac{1}{2} f'(l_j) + x_T .$$

If $ah \geqslant \epsilon$, compute

$$y_1 = a/(M_2 - M_1), \qquad x_1 = h - b/a - \tfrac{1}{2}y_1 ,$$

$$c = f'(l_j) + M_1 x_1 ,$$

then if $c \geqslant 0$ and either $M_1 \geqslant 0$ or $f'(l_j) \geqslant 0$ go to (*).

If $c \geqslant 0$ otherwise, put $x_T = -f'(l_j)/M_1$, $f_T = f(l_j) + \tfrac{1}{2}f'(l_j)x_T$.
If $c \leqslant -M_2 y_1$ and either $M_1 \leqslant 0$ or $f''(u) \leqslant 0$ go to (*).
If $c \leqslant -M_2 y_1$ otherwise, put

$$x_T = u - f'(u)/M_1, \qquad f_T = f(u) - \tfrac{1}{2}\{f'(u)\}/M_1 .$$

If $-M_2 y_1 < c < 0$, put

$$x_T = x_1 - c/M_2, \qquad f_T = f(l_j) + \tfrac{1}{2}\{f'(l_j)x_1 + cx_T\} .$$

If $f_T > f_{\min} - \epsilon$, go to (*).
Otherwise put $j = j + 1$, $l_j = l_{j-1} + x_T$ and compute $f(l_j)$, $f'(l_j)$ and $a_i''(l_j)$. If $f(l_j) < f_{\min}$, put $f_{\min} = f(l_j)$ and $x_{\min} = l_j$.
If $M_1 < 0$, put $I_j = 0$ and start a new step of the algorithm.
If $M_1 \geqslant 0$ and $f'(l_j) \geqslant 0$ go to (*).
If $M_1 \geqslant 0$ and $f'(l_j) < 0$, put $I_j = 1$; and if I_{j-1} put $l_{j-1} = l_j$, $f(l_{j-1}) = f(l_j)$, $a_i''(l_{j-1}) = a_i''(l_j)$ and $j = j - 1$
Start a new step of the algorithm.

References

[1] E.M.L. Beale and J.A. Tomlin, "Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables", in: J. Lawrence, ed., *Proceedings of the fifth international conference on operational research* (Tavistock Publications, London, 1970) pp. 447–454.

[2] J.E. Falk and R.M. Soland, "An algorithm for separable nonconvex programming problems", *Management Science* 15 (1969) 550–569.

[3] J.J.H. Forrest, J.P.H. Hirst and J.A. Tomlin, "Practical solution of large and complex integer programming problems with UMPIRE", *Management Science* 20 (1974) 736–773.

[4] G.P. McCormick, "Attempts to calculate global solutions of problems that may have local minima", in: F.A. Lootsma, ed., *Numerical methods for non-linear optimization* (Academic Press, New York, 1972) pp. 209–221.

[5] C.E. Miller, "The simplex method for local separable programming", in: R.L. Graves and P. Wolfe, eds., *Recent advances in mathematical programming* (McGraw Hill, New York, 1963) pp. 69–100.

[6] R.M. Soland, "An algorithm for separable nonconvex programming problems II: nonconvex constraints", *Management Science* 17 (1971) 759–773.