FILOMAT (Niš) **12:1** (1998), 1–8

SYMBOLIC IMPLEMENTATION OF LEXICOGRAPHIC MULTICRITERIA PROBLEMS

Predrag S. Stanimirović and Svetozar Rančić

Abstract. We describe implementation of lexicographic multicriteria problems by means of symbolic processing available in the programming language SCHEME.

1. Introduction

The optimization methods are implemented in procedural programming languages, mainly in FORTRAN [1], [7].

In the programming package MATHEMATICA [8], [9] are available a few functions for numerical optimization. The function *FindMinimum* finds local minimum for a given function and starting point.

The functions *ConstrainedMin* and *ConstrainedMax* allow you to specify a linear objective function to minimize or maximize, together with a set of linear inequality constrains on variables. In all cases is assumed that the variables are constrained to have non-negative values. More precisely, only the linear programming is implemented in MATHEMATICA.

`ConstrainedMin[f, {inequalities}, {x, y, ...}]` find the global minimum of f, in the region specified by *inequalities*;

`ConstrainedMax[f, {inequalities}, {x, y, ...}]` find the global maximum of f, in the region specified by *inequalities*.

In papers [5], [6] we describe implementation of the search methods and first order gradient optimization methods in LISP environment.

In this paper we investigate minimization of a given ordered sequence of real objective functions and given set of inequality constrains:

$$\text{(L)} \qquad \begin{aligned} &\text{Minimize: } Q_1(\vec{x}), \ldots, Q_l(\vec{x}), \quad \vec{x} \in \mathbb{R}^n \\ &\text{Subject to: } F: \quad f_i(\vec{x}) \leq 0, \quad i = 1, \ldots, p \end{aligned}$$

Received July 15, 1997

1991 *Mathematics Subject Classification*: 90C30, 68N15.

Key words and phrases. Lexicographic multicriteria problems, constrained minimization, SCHEME.

in the programming language SCHEME.

It is well known that the stated problem (L) is equivalent to the following sequence of convex constrained nonlinear programming problems [10]:

$$
\begin{aligned}
&\text{Minimize: } Q_k(\vec{x}), \quad \vec{x} \in \mathbb{R}^n \\
&\text{Subject to: } Q_{k-1}(\vec{x}) \leq a_{k-1}, \\
&\qquad \cdots\cdots \\
&\qquad Q_1(\vec{x}) \leq a_1, \\
&\qquad \vec{x} \in F, \qquad\qquad k = 1, \ldots, l
\end{aligned}
\tag{Lk}
$$

where a_i, $i = 1, \ldots, k-1$ are the optimal values for the before stated problems (Li), $i = 1, \ldots, k-1$.

Our main goal is to describe several implementation details of the lexicographic multicriteria problems which are specific for the symbolic manipulation in the programming language SCHEME. SCHEME [4], dialect of the programming language LISP, is one of the most versatile programming languages available today, applicable in both symbolic and numeric processing.

We suggest the following advantages arising from the implementation of the lexicographic multicriteria problems using the possibility of the symbolic processing in LISP environment:

1. Possibility of any implementation procedure to take an arbitrary objective function (which is not defined by a subroutine) and given constrains. Corresponding parameters are incorporated into the convenient LISP's *internal form*, which is used as a formal parameter.
2. Possibility for moving the selected objective functions and functions forming given constrains, from the list representing the internal form of the stated problem, into the formal parameter list of the procedure implementing constrained optimization methods.
3. Simple construction of the internal form representing the constrained program (Lk). This is implied by the LISP's functions for list manipulation, which allow you to prepend an inequality constrain $Q_i(\vec{x}) \leq a_i$, $1 \leq i \leq l$ into the list of inequality constrains
4. LISP allows you to use arrays of functions, whose elements can be selected and later applied to supplied arguments or used as first-class data objects. Such structures are not convenient in procedural programming languages.

A numerical result is reported and relations with the corresponding results, obtained by means of the procedural programming languages, are discussed.

2. Implementation details

In the paper [3] we introduce the *internal form* appropriate for nonlinear constrained problems. The nonlinear constrained problem

$$\begin{aligned} &\text{Minimize: } Q(\vec{x}), \quad \vec{x} \in \mathbb{R}^n \\ &\text{Subject to: } f_i(\vec{x}) \leq 0, \quad i \in \mathcal{P} = \{0, \ldots, p\} \\ &\qquad\qquad h_j(\vec{x}) = 0, \quad j \in \mathcal{Q} = \{0, \ldots, q\}. \end{aligned}$$

is transformed in the following LISP's form:

```
        '(   Q(x)      (x)
             ( f_1(x) ··· f_p(x) )
(I)
             ( h_1(x) ··· h_q(x) )
         )
```

If one of the inequality or equality constrains is absent, the corresponding list is empty.

The internal form, denoted by q, of the starting problem (L) is the following list:

```
'( ( Q_1(x) ··· Q_l(x) )
   (x)
   ( f_1(x) ··· f_p(x) )
   ()

)
```

By the expression `(car q)` we select the list of the objective functions, the expression `(cadr q)` contains the list of independent variables used in the objective function, and `(caddr q)` contains list of the functions which define the given constrains. In this way, we describe the advantage 1.

The vector containing the objective functions can be formed by means of the expression:

```
(set!  vecfun (list->vector (car q)))
```

We are now in a position to take elements from the vector $vecfun$, as the functions applicable to supplied arguments, and which can be used as arguments in LISP's functionals.

A structure representing a vector whose elements are functions can not be easy implemented in procedural programming languages. Also, an arbitrary

ith element from the constructed vector $vecfun$ can be transformed into the corresponding lambda-function:

```
(set!  fun (list 'lambda (cadr q) (vector-ref vecfun i))))
```

It is assumed that all objective functions use the same list of parameters.

This represents the advantage 4.

Moreover, the functions selected from the vector $vecfun$ can be incorporated in symbolic expressions which will be used as the internal form of the corresponding constrained nonlinear program, in procedures implementing constrained minimization problems. The functions which implement constrained nonlinear programs (exterior and interior point methods, Rockafellar's method) are described in [3].

The list of functions, representing given inequality constrains F is extracted, from the internal form q, by the expression

```
(set!  ineq (caddr q) )
```

The internal form of the problem (Lk), obtained in the kth step $1 \leq k \leq l$, which possesses the form (I), applicable in the routines which performs the constrained nonlinear optimization methods (for example the Rockafellar's method), described in [3], is the following list:

$$
\begin{array}{l}
\texttt{'(}\ Q_k(\vec{x}) \qquad (\vec{x}) \\
\quad (\ (-\ \ Q_{k-1}(\vec{x})\ \ a_{k-1}) \\
\qquad \ldots\ \ldots \\
\qquad (-\ \ Q_1(\vec{x})\ \ a_1) \\
\qquad f_1(\vec{x}) \\
\qquad \ldots\ \ldots \\
\qquad f_p(\vec{x}) \\
\quad) \\
\quad () \\
)
\end{array}
$$

The internal form, denoted $funpom$, of this nonlinear constrained program can be formed as follows:

Step 1. Append the empty list in the list of inequality constrains:

```
(set!  ineq (list ineq nil))
```

Step 2. Form the internal form of the constrained problem (Li), $1 \leq i \leq l$.

```
(set!  q (vector-ref vecfun i))
(set!  funpom (cons q (cons (cadr q) ineq)))
```

This is description of the advantage 2.

A feasible starting point x and the precision *eps* can be selected by the expressions

`(set! x (read))` and `(set! eps (read))`.

The main cycle is defined as follows:

```
(do ((i 0))
   ; If all of the objective functions are minimized
   ; return the list res,
   ; containing minimal value and optimal point
   ((= i (length (car q)) res)
   ; Select the ith objective function
   (set! q (vector-ref vecfun i))
   ; Form the internal form of the constrained problem Li
   (set! funpom (cons q (cons (cadr q) ineq)))
   ; Solve ith constrained problem, using the Rockafellar's
   ;  method [3], called by the function rok
   ; the result res is the list (x*, a_i), a_i = Q_i(x*)
   (set! res (rok funpom x eps))
   ; Prepend the new constrain Q_i(x) - a_i <= 0
   (set! fun (list '- (vector-ref vecfun i) (cadr res)))
   (set! ineq (cons (cons fun (car ineq)) (cdr ineq)))
   ; Increment the parameter
   (set! i (+ i 1)))
)
```

The essence of advantage 3 is contained in the following code:

Step 1. To form the function $Q_i(\vec{x}) - a_i$, representing the inequality constrain $Q_i(\vec{x}) - a_i \leq 0$, where $res = (\vec{x}^*, a_i)$, $a_i = Q_i(\vec{x}^*)$ is returned from the function *rok*, implementing before stated constrained program, of the form (Lk), $k = i - 1$:

```
(set! fun (list '- (vector-ref vecfun i) (cadr res)))
```

Step 2. Prepend the expression $(- \quad a_i \quad Q_i(\vec{x}))$, which represents the function contained in the constrain $Q_i(\vec{x}) - a_i \leq 0$, into the list of constrains:

```
(set! ineq (cons (cons fun (car ineq)) (cdr ineq)))
```

In the above routine we construct an extensible list whose starting value, defined in the symbol *ineq*, contains the functions used in inequality constrains. This list is modified by prepending the list $(- \quad Q_i(\vec{x}) \quad a_i)$ in ith iteration, $i = 1, \ldots, l$:

`(set! ineq (cons (cons fun (car ineq)) (cdr ineq)))`
This list is used as an element of the internal form representing the generated constrained problem Li, $1 \leq i \leq l$, which is supplied to an arbitrary procedure which implement constrained optimization methods.

3. Examples

Example 3.1. Consider the following optimization problem

$$\begin{aligned} \text{Minimize: } & \{Q_1(\vec{x}) = -x_1 - x_2, \quad Q_2(\vec{x}) = -x_2^2 - 4x_2\} \\ \text{Subject to: } & -x_1 \leq 0, \\ & -x_2 \leq 0, \\ & -x_3 \leq 0, \\ & x_1 + x_2 + x_3 - 1 \leq 0. \end{aligned}$$

The internal form of the presented problem is the list

```
'(
  ( (-  0  (+  x1  x2))
    (-  (*  x2  x2)  (*  4  x2))
  )
  (x1  x2  x3)
  ( (-  0  x1) (-  0  x2) (-  0  x3)(-  (+  x1  x2  x3) 1) )
)
```

Applying the above defined methods with the accuracy $1.e-8$, and starting from the point (1 0 0), we get the following numerical results:
The optimal value of the constrained problem

$$\begin{aligned} \text{Minimize: } & Q_1(\vec{x}) = -x_1 - x_2 \\ \text{Subject to: } & -x_1 \leq 0, \\ & -x_2 \leq 0, \\ & -x_3 \leq 0, \\ & x_1 + x_2 + x_3 - 1 \leq 0. \end{aligned}$$

is (0.999999999966572 $-3.34278198440878e{-}11$ $-3.34278198440878e{-}11$) and the optimal value is $a_1 = -0.999999999933144$.
In the second iteration, the optimal point of the constrained problem

$$\begin{aligned}
\text{Minimize: } & Q_2(\vec{x}) = -x_2^2 - 4\,x_2 \\
\text{Subject to: } & -x_1 - x_2 \leq a_1 \\
& -x_1 \leq 0, \\
& -x_2 \leq 0, \\
& -x_3 \leq 0, \\
& x_1 + x_2 + x_3 - 1 \leq 0.
\end{aligned}$$

is $(-6.75086975150132e{-}10 \quad 1.00000000179395 \quad -5.16342917849918e{-}10)$ and the optimal value is -3.00000000358791.

Note that the exact solution of the optimal point is the vector $(0 \quad 1 \quad 0)$.

In [10] is obtained similar result, with the precision 10^{-6}. This problem in MATHEMATICA can not be solved.

4. Conclusion

Our main goal is a symbolic implementation of the multiobjective lexicographic optimization methods. As far, as we know, these methods are implemented only in procedural programming languages. Also, as a motivation for the symbolic implementation is the absent of nonlinear multiobjective optimization in functional programming languages. The improvements are ensured primarily applying possibility of symbolic processing of the functional programming languages PC SCHEME. Of course, similar principles are valid for the other functional programming languages. But, we prefer PC SCHEME because of their ability in the symbolic processing as well as in the numeric processing. The main purpose is to point out that the proper selection of the programming language in multiobjective optimization is a language applicable in symbolic processing as well as in numerical computations.

References

[1] David, M.H., *Applied Nonlinear Programming*, McGraw-Hill Book Company, 1972.

[2] Henessey, L.W., *Common* LISP, McGraw-Hill Book Company, 1989.

[3] Rančić, S., *Application of the language LISP in mathematical programming*, Master thesis, University of Niš, 1997. (In Serbian)

[4] Smith, J.D., *An Introduction to Scheme*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[5] Stanimirović, P. and Rančić, S., *Unidimensional search optimization in LISP*, Proceedings of the II Mathematical Conference in Priština (1996), 253–262.

[6] Stanimirović, P. and Rančić, S., *Unconstrained optimization in LISP*, in: XI Conference on Applied Mathematics, Budva (1996), 355–362.

[7] Stojanov, S., *Methods and Algorithms for Optimization*, Drzavno izdatelstvo, Tehnika, Sofija, 1990. (In Bulgarian)

[8] Wolfram, S., *Mathematica: a system for doing mathematics by computer*, Addison-Wesley Publishing Co, Redwood City, California, 1991.

[9] Wolfram, S., *Mathematica Book, Version 3.0*, Wolfram Media and Cambridge University Press, 1996.

[10] Zlobec, S. and Petrić, J., *Nonlinear programming, Naučna Knjiga, Beograd, 1989.* (In Serbian)

University of Niš, Department of Mathematics Ćirila i Metodija 2, 18000 Niš Yugoslavia
E-mail: pecko@archimed.filfak.ni.ac.yu