MOPEN: A computational package for Linear Multiobjective and Goal Programming problems

R. Caballero*, M. Luque, J. Molina, F. Ruiz

Department of Applied Economics (Mathematics), Faculty of Economics, University of Málaga, Campus El Ejido s/n. Málaga, 29071, Spain

Received 30 November 2002; accepted 1 June 2004
Available online 14 August 2004

Abstract

MOPEN is a computational package designed as a global tool for Linear Multiobjective and Goal Programming problems with continuous and/or integer variables. The main existing techniques for these problems have been included in this package. That is, it is possible to generate or approximate the efficient set using Generating Methods, to obtain Compromise solutions or to use Goal Programming or reference Point approaches. As will be described, many advanced options have been implemented with every method. MOPEN has been implemented under a Windows environment; thus, it is easy to build and handle the data entry files and the result layout files. The behavior of MOPEN—in terms of CPU time used to solve large problems—can be considered as good; therefore, this package is a powerful tool to handle the previously mentioned problems.

Keywords: Multiobjective Programming; Computational package; Goal Programming; Integer programming

1. Introduction

1.1. Basic Definitions

In this paper, the general Linear Multiobjective problem

\[
\begin{align*}
\text{Min}(f_1(x), f_2(x), \ldots, f_p(x)) &= \left(c_1^T x, c_2^T x, \ldots, c_p^T x\right) \\
\text{s.t.:} \quad x &\in X = \{x \in \mathbb{R}^n : Ax \leq b\} \\
A &\in M_{p \times n}(\mathbb{R}), \quad b \in \mathbb{R}^p
\end{align*}
\]

(LMOP)

will be considered. The following classical concepts will be used throughout the paper:

- Let \(x_i^*\) be the optimum value of \(f_i\): \(f_i(x_i^*) = \min \{f_i(x)/x \in \mathbb{R}^n, A \cdot x \leq b\}\). \(x_i^*\) is called the Ideal Solution of \(f_i\), and \(f_i^* = f(x_i^*)\) is its Ideal Value.
- The pay-off matrix is formed by the values of all the functions \(f_i\) in all the ideal solutions:

\[
\begin{bmatrix}
 f_1(x_1^*) & f_1(x_2^*) & \cdots & f_1(x_n^*) \\
f_2(x_1^*) & f_2(x_2^*) & \cdots & f_2(x_n^*) \\
\vdots & \vdots & \ddots & \vdots \\
f_n(x_1^*) & f_n(x_2^*) & \cdots & f_n(x_n^*)
\end{bmatrix}
\]
The elements of the main diagonal of the payoff matrix are the ideal values of each function. The maximum value per column, $\max_{j=1, \ldots, p} \{f_i(x_j)\}$, is called the Anti-ideal Value of $f_i$ and is denoted by $f_i^{(0)}$. The corresponding solution is called the Anti-ideal Solution of $f_i$ and is denoted by $x_i^{(0)}$.

- A feasible solution $x^* \in X$ is said to be efficient for (LMOP) (or Pareto optimal) if there does not exist any other solution $x \in X$, such that:
  \[ f_i(x) \leq f_i(x^*) \quad \forall i = 1, \ldots, p \]
  with at least one $j \in \{1, \ldots, p\}$ such that $f_j(x) < f_j(x^*)$.

- A feasible solution $x^* \in X$ is said to be weakly efficient for (LMOP) (or weakly Pareto optimal) if there does not exist any other solution $x \in X$, such that:
  \[ f_i(x) < f_i(x^*) \quad \forall i = 1, \ldots, p. \]

The most widely accepted classification of the existing techniques to solve these problems depends on the information flow between the decision maker and the analyst. The first possibility is that the decision maker provides his/her preferences only by stating which objectives have to be minimized and which ones have to be maximized. In this case, the aim of the methods is to show the Pareto efficient set (or an approximation of it). The techniques corresponding to this scheme are called Generating Techniques (see, for example, Ref. [16]). Second, the decision maker may want to overcome the conflict among the objectives, without having to state a clear preference towards a specific one. In this case, an efficient solution has to be found, characterized by creating a compromise or equilibrium among the objectives. This is the basic idea of Compromise Programming [23,24,15]. On the other hand, prior to the resolution process, the decision maker provides information in the form of target values, $\alpha_i$, for each objective, and possibly preference levels among them, then this constitutes the Goal Programming scheme (see Refs. [5,6,8]). Similarly, the Reference Point method [22] lets the decision maker establish aspiration levels for the objectives, without having to renounce the efficiency of the solutions. Finally, in the Interactive Techniques (see, for example, [9]), there exists a continuous flow of information between the analyst and the decision maker, throughout the whole resolution process).

In this paper, the software MOPEN is described, in which the main algorithms belonging to the two first groups have been implemented, while the interactive techniques are implemented in PROMOIN (see Ref. [4]).

1.2. Existing implementations

Two main implementations can be found within the previously described framework: ADBASE and the systems GPSYS and IGPSYS. ADBASE was developed by Ralph Steuer in 1974, in the FORTRAN language, although, since then, it has undergone several revisions and improvements. This software determines all the efficient vertices and edges of a Linear Multiobjective problem. Besides this, it also has the option to solve a Lexicographic Goal Programming problem. This program runs under an MSDOS environment and uses a specific format for the data entry files. The original idea of the author was to create a code that was general enough to be implemented under any operating system, in a personal computer, and at a time when the use of Windows was not very widespread. These data entry files can only be edited by an MSDOS program, and this fact can be inconvenient when using Windows. On the other hand, GPSYS and IGPSYS, developed by M. Tamiz and D.F. Jones, are practically the only implementations for Goal Programming problems available at the moment. GPSYS solves Linear Goal Programming problems, and IGPSYS solves Integer Linear Goal Programming problems. Both programs have been implemented in the FORTRAN language and run under MSDOS. The implementations include all the normalization possibilities for Goal Programming as well as several options to detect and restore the efficiency of the final solutions. Nevertheless, the difficulties encountered regarding editing the data entry file are similar to those described for ADBASE. Thus, both implementations are highly efficient from a computational point of view, although they do not take advantage of the benefits offered by the implementation under a Windows environment.
Within the MOP implementations under Windows, the TOMMIX package [1] can be found in the literature. This software includes some of the most widely used Interactive MOP methods (STEM, Ziont–Wallenius, Interval Criterion Weights and Pareto Race methods) as well as their own method, TRIMAP. This package takes advantages of the Windows environment, but is mainly focused on Interactive methods. Apart from this specific software, all the main LP solvers can be used to obtain efficient solutions or to solve Goal Programming approaches. Obviously, these programs do not make a thorough analysis from the multicriteria point of view.

The implementation described in this paper tries to fulfill the need for systems for Multiobjective and Goal Programming problems under a Windows environment so that the data entry and layout format is simple and intuitive.

2. MOPEN

2.1. Technical features

MOPEN is a program for Windows, implemented in the C++ language; that is, the compiler Microsoft Visual C++, version 6.0, has been used. Presently, two versions of MOPEN are available. MOPEN 1.1 uses as the libraries NAG in C (see Ref. [10]), mark 4, CPLEX version 6.5.1 (see Ref. [7]) and NAG in FORTRAN, mark 18 (see Ref. [11]) to solve the single criterion linear problems. MOPEN 1.2 has its own incorporated subroutines to solve such problems. The advantage of the former version is its quickness, while the latter is slower but does not require any license to use the libraries. In these current versions, the program can only solve linear problems, more precisely problems with up to 15,000 variables, 1500 constraints and 100 objective functions can be solved. Both versions are available upon request for all the researchers who wish to have a copy.

The performance of MOPEN, in terms of CPU time, is shown in Table 1, where the times for a series of test problems appear.

2.2. Working environment

Fig. 1 shows the main window frame of MOPEN, where the resolution, weight determination, options and help menus can be observed. Inside the main frame, the file-editing zone can also be seen. This zone makes it very simple to manage and visualize the information throughout the resolution process. The format of these files, as previously mentioned, is simple and intuitive, and they are edited in the same way as any text file in

<table>
<thead>
<tr>
<th>Problem</th>
<th>Variables</th>
<th>Constraints</th>
<th>Functions</th>
<th>Goals</th>
<th>P levels</th>
<th>Type</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>–</td>
<td>–</td>
<td>Cont.</td>
<td>7.64 s</td>
</tr>
<tr>
<td>#2</td>
<td>5</td>
<td>5</td>
<td>–</td>
<td>4</td>
<td>2</td>
<td>Cont.</td>
<td>0.31 s</td>
</tr>
<tr>
<td>#3</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>Int.</td>
<td>27.36 s</td>
</tr>
<tr>
<td>#4</td>
<td>6</td>
<td>7</td>
<td>–</td>
<td>4</td>
<td>2</td>
<td>Int.</td>
<td>8.68 s</td>
</tr>
<tr>
<td>#5</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>Cont.</td>
<td>1.74 s</td>
</tr>
<tr>
<td>#6</td>
<td>8</td>
<td>10</td>
<td>–</td>
<td>3</td>
<td>1</td>
<td>Cont.</td>
<td>2.38 s</td>
</tr>
<tr>
<td>#7</td>
<td>10</td>
<td>12</td>
<td>4</td>
<td>–</td>
<td>–</td>
<td>Cont.</td>
<td>9.93 s</td>
</tr>
<tr>
<td>#8</td>
<td>128</td>
<td>100</td>
<td>20</td>
<td>–</td>
<td>–</td>
<td>Cont.</td>
<td>21.03 s</td>
</tr>
<tr>
<td>#9</td>
<td>128</td>
<td>84</td>
<td>–</td>
<td>48</td>
<td>6</td>
<td>Cont.</td>
<td>0.98 s</td>
</tr>
<tr>
<td>#10</td>
<td>140</td>
<td>36</td>
<td>–</td>
<td>140</td>
<td>3</td>
<td>Cont.</td>
<td>0.96 s</td>
</tr>
<tr>
<td>#11</td>
<td>252</td>
<td>171</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>Cont.</td>
<td>3.25 s</td>
</tr>
<tr>
<td>#12</td>
<td>252</td>
<td>171</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>Int.</td>
<td>2 min 46 s</td>
</tr>
<tr>
<td>#13</td>
<td>3,124</td>
<td>712</td>
<td>3</td>
<td>710</td>
<td>5</td>
<td>Int.</td>
<td>3 h 46 min</td>
</tr>
</tbody>
</table>

The columns indicate the problem number, number of variables, number of constraints, number of objective functions, number of goals, number of priority levels, type of variables (continuous or integer) and overall computing time. The computing time is the mean time required in five randomly generated problems of the same size.
Windows. Finally, MOPEN has a complete help system, where all the details concerning the functioning of the program can be found. In addition, information regarding the creation of data files, details about the algorithms implemented, normalization options, efficiency tests, etc., appear in the help system. All the available methods can be applied to a problem whose corresponding data are stored in a single file.

All the Multiobjective Programming algorithms implemented in the program can be found in the resolution menu. Many methods require the decision maker to give local weights for the objective functions and/or goals. As this may not be an easy task, MOPEN offers, in the ‘Weight Assistant’ menu, a series of options devoted to facilitate this weight determination process. The user can generate weights by direct assignment methods (Ordering, Simple Assessing and Ratio Comparison) and by pairwise comparison methods (AHP, Geometric Mean, Pseudo-Inverse and Goal Programming). All the details concerning these weight determination methods can be found in Ref. [13]. The menu offers the possibility of including these weights in the file corresponding to the problem.

### 2.3. Data entry

MOPEN makes it possible to include in a single file all the data regarding the objective functions, goals and general features of the problem. This file can be created using a template (New File) or using an assistant program (Use Assistant). For small problems, the use of the assistant is recommended because it allows the user to provide all the data through a series of windows in an intuitive way. On the other hand, for large problems, it will generally be more comfortable to use the template because this option makes it possible to import in an easy way data that have been previously stored in other files, e.g., spreadsheets.

For example, let us consider the following problem (P):

- **Objective functions** (for the generating techniques):
  
  \[
  \begin{align*}
  \max f(x, y, z, t, u, v) &= 4x + 3y + z - 3t + 2u \\
  \min g(x, y, z, t, u, v) &= 3x + 2y + 6z - 2t + v \\
  \min h(x, y, z, t, u, v) &= x + y + z - t + u - v 
  \end{align*}
  \]
– Goals (for the Goal Programming option):

| Priority | Level 1 | \( G_i^f: 4x+3y+z-3t+2u+n_1-p_1=25 \) (min \( n_1 \))
| Priority | Level 2 | \( G_i^g: 3x+2y+6z-2i+v+n_2-p_2=11 \) (min \( p_2 \))
| Priority | Level 3 | \( G_i^h: x+y+z-t+u-v+n_3-p_3=10 \) (min \( p_3 \))

– Constraints:
\[
\begin{align*}
x + y - z + 3t + u + 2v & \leq 15 \\
2x + y + 3z - t + 4u + v & \leq 24 \\
x, y, z, t, u, v & \geq 0 \\
x, y, z, t, u, v & \in \mathbb{R}
\end{align*}
\]

The data file corresponding to problem \((P)\) will be called Sample.txt and is displayed in Fig. 2.

Let us observe that in this file, each group of data is labeled. Thus, lines such as Problem_Name, Number_of_variables or Number_of_Objectives appear in the file. These lines allow the program to identify the data, and any modification of such labels would cause a reading error. In order to prevent the user from modifying these lines, the template already contains them so that the user will only have to introduce the data.

2.4. Resolution

In order to solve the Multiobjective problems, MOPEN offers five groups of resolution algorithms. First, there are the efficient set generating techniques. In this group, the user can find the Weighting Method which, apart from introducing the weights during run time, has the option to automatically generate a set of weights in order to obtain an approximation of the efficient set with as many points as desired. The weight generating process is as follows:

Step 1 The user is asked to give the number \( q \) of weights for each objective (apart from the 0 weight).
Step 2 Function 1: from \( i_1=q \) to \( i_1=0 \) (step \(-1\)).
Function 2: from \( i_2=q-i_1 \) to \( i_2=0 \) (step \(-1\)),

\[
\lambda_j = \frac{i_j}{q},
\]

Finally, the vector of weights \((\lambda_1, \lambda_2, \ldots, \lambda_p)\) is normalized according to the scheme chosen by the user (see Section 2.5).

It must be taken into account that if some zero weight is given, then the corresponding solution is only guaranteed to be weakly efficient. In this situation, MOPEN carries out an efficiency test of the solution. Namely, if \(x^*\) is the solution to be tested, the following \( p \) constraint problems are solved:

\[
(C_j) \quad \begin{cases}
\min & f_i(x) \\
\text{s.t.} & x \in X \\
& f_j(x) \leq f_i(x^*) \quad (i = 1; \ldots; p; \ i \neq j)
\end{cases}
\]

If \(x^*\) is an optimal solution for all the problems, then it is guaranteed that it is an efficient solution of \((P)\). If not, \(x^*\) is just weakly efficient (and at least an efficient solution that dominates it is found in this process).

The Constraint Method can also be found in this group. Analogously to the previous case, the constraints can be either introduced by the user during run time or automatically generated by the program in order to approximate the efficient set. In this case, the bounds generation process is as follows:

Step 1 The user is asked to give the number \( q \) of bounds for each objective between its ideal and its anti-ideal values \((q \geq 2)\).
Step 2 From \( i=1 \) to \( p \). Function 1: from \( k_1=i_1^* \) to \( i^{(1)} \) step \( \frac{1}{q} \) \((i^{(1)} - i_1^* ) \ldots \)
(excluding function \( i \)).
Function \( p \): from \( k_p=i_p^* \) to \( i^{(p)} \) step \( \frac{1}{q} \) \((i^{(p)} - i_p^* ) \ldots \)
Given the vector \((i_1, i_2, \ldots, i_{p-1})\), let:
\[
\lambda_j = \frac{i_j}{q},
\]

The following constraint problem is generated:

\[
(C_j) \quad \begin{cases}
\min & f_i(x) \\
\text{s.t.} & x \in X \\
& f_j(x) \leq k_j \quad (j = 1; \ldots; p; \ j \neq i)
\end{cases}
\]

The second group is related to Compromise Programming [23]. More precisely, in this option, for a given vector of weights, the \( L_1 \) and \( L_\infty \) compromise solutions associated with the multi-criteria problem under study are obtained. These are only two of the infinite compromise solutions that can be obtained (one for each metric \( L_p \)). Only in the case when there are only two objectives do these two solutions define the compromise set.
(which is the segment that joins them). Again, an option is offered to automatically generate sets of weights (following the same procedure used for the weighting method) so as to generate efficient solutions for the problem.

The third group incorporates all the Goal Programming techniques. Thus, any of the main schemes of this field can be applied, namely, the Lexicographic, Minimax, Weighted, Extended and Interval options are available. The differences among these options are the
achievement functions considered in each of them. Let the goals have the form \( f_i(x) + n_i - p_i = z_i, \) then the three most common achievement functions are functions of the undesired deviation variables. Let \( I_k \) denotes the set of indexes corresponding to the goals placed in the \( k \)th priority level, whose undesired deviation variable is \( p \). Similarly, \( J_k \) is defined for the goals whose undesired deviation variable is \( n \), and \( K_k \) for \( n+p \) (if there is just one level, the subscript will not be used).

- **Weighted scheme:**
  
  \[
  \min H(n,p) = \sum_{i \in I} \mu_i p_i + \sum_{i \in J} \mu_i n_i + \sum_{i \in K} \mu_i (p_i + n_i),
  \]

- **Minmax scheme:**
  
  \[
  \min H(n,p) = \max_j \{ \mu_j p_j \} + \max_{j \in J} \{ \mu_j n_j \} + \max_{i \in K} \{ \mu_i (p_i + n_i) \},
  \]

- **Lexicographic scheme:**
  
  \[
  \text{lexmin} \ (h_1(n,p), h_2(n,p), \ldots, h_s(n,p))
  \]

- **The Extended Lexicographic Goal Programming approach** is a compromise scheme between the weighted and the minmax ones:
  
  \[
  \begin{align*}
  \text{lexmin} & \ (\hat{\lambda}_1 d_1 + (1 - \hat{\lambda}_1) h_1(n,p), \ldots, \hat{\lambda}_s d_s + (1 - \hat{\lambda}_s) h_s(n,p)) \\
  \text{s.t.} & \ x \in X \\
  & \ f_i(x) + n_i - p_i = z_i, \ i = 1, \ldots, p \\
  & \mu_i^k p_i \leq d_k, \ i \in I_k, \ k = 1, \ldots, s \\
  & \mu_i^k n_i \leq d_k, \ i \in J_k, \ k = 1, \ldots, s \\
  & \mu_i^k (p_i + n_i) \leq d_k, \ i \in K_k, \ k = 1, \ldots, s
  \end{align*}
  \]

- Finally, the Interval Goal Programming scheme allows the use of the so-called U-Penalty penalizing functions for the unachievements of the goals. In these functions, the user can establish different thresholds so that different unachievement levels are given different importance. For further details, see Refs. [17] and [19].

Once the Goal Programming resolution has been carried out, following any of the previously mentioned techniques, MOPEN allows the user to use several techniques of efficiency restoration of the solution. The efficiency restoration can be carried out using the Direct, Preference Based, Interactive or G.S.I. (see Ref. [3]) methods. All the details about Goal Programming and Restoration methods can be found in Ref. [18] for the continuous case and Ref. [21] for integer problems.

A new approach can be found in the fourth group: Satisfying and Efficient Solutions (see Ref. [2]). Under this approach, if the Goal Programming problem has solutions that satisfy all the goals, then efficient solutions are calculated within the satisfying set, i.e., within the set \( S = \{ x \in X | f_i(x) \leq \alpha_i, \ i = 1, \ldots, p \} \). To this end, the Weighting Method and the Constraint Method can be used, where families of weights or bounds, respectively, are automatically generated as previously described for the generating techniques. The efficiency of the solutions may be determined with respect to a set of objectives, which may or may not be related to the goals (see Section 2.6).

Finally, the fifth option corresponds to the Reference Point Method (see Ref. [22]). This method uses the Tchebychev achievement function in order to obtain, for a given set of target values and a given vector of weights, a satisfying and efficient solution. Again, the weights can be automatically generated in order to obtain a good approximation of the set of efficient and satisfying solutions.

The menu corresponding to these five groups of methodologies can be seen in Fig. 3.

Once the data file has been selected, a window appears in the screen, which shows the default names of all the solution files where the solutions corresponding to the current problem will be stored. Generally, the names of such files are assigned taking into account the problem name and the resolution method:

```
Sol_<Resolución Method>_<File Name/Problem Name>.txt
```

Finally, after solving the problem using a certain algorithm, the user will be offered the option of filtering the solutions obtained. This possibility will
be described in Section 2.5. The default name of the file where the filtered solutions are stored is Fil_<Name of the corresponding Solution File>. All these default names can be changed at any moment during the resolution process.

### 2.5. Utilities

The **Options** menu of MOPEN offers a series of tools in order to make it easier for the user to solve a problem, complete the data of a problem, choose normalization options, visualize data, etc. Fig. 4 shows these options.

First, let us highlight the **Obtain Ideal and Anti-Ideal Values** option. It can be very useful in many cases to know the ideal and anti-ideal values corresponding to each goal and/or objective, before starting the resolution process. In the case of a goal, it will be assumed that the corresponding function is to be maximized if the goal takes the form $f_i(x) \geq z_i$ and to be minimized if the goal takes the form $f_i(x) \leq z_i$. In general, these values can be helpful when fixing target values, choosing a function to be optimized, choosing the bounds in the weighting method or selecting a normalizing option. The **Filtering** option can also be found in this menu. First, the user is asked whether he/she prefers to filter using the values of the decision variables or the values of the objective functions. After this, the filtering radio $r$ must be provided, which determines how fine the filtering process will be. Let us assume that the user decides to filter the results taking into account the values of the decision variables and let $\{x_1, x_2, \ldots, x_n\}$ be the list of solutions to be filtered. Then the process is as follows:

**Step 0** Let $X_f=\emptyset$. Let $x_r=x_n$ be the reference point. Let $X_1=\{x_1, x_2, \ldots, x_{n-1}\}$. Let $k=0$.

**Step 1** Let $k=k+1$. $X_f=X_f \cup \{x_r\}$. $A=\emptyset$.

For all $x_i \in X_{k-1}$, if $\sum_{j=1}^{n} |x_j^i - x_j^r| \leq r$, then make $A=A \cup \{x_i\}$.

**Step 2** Let $X_k=X_{k-1} \setminus A$.

**Step 3** If $X_k=\emptyset$, then go to Step 5.

**Step 4** Let $x_r$ be the point belonging to $X_k$ whose subindex is the greatest. Go to Step 1.

**Step 5** End of the filtering procedure. The set of filtered solutions, $X_f$, is transferred to the file Fil_<Name of the corresponding Solution File>.

The **Branch and Bound Options** of MOPEN offers a series of strategies for the application of the Branch and Bound method to integer problems. The user has the possibility to choose among the many different options of the method. This fact gives MOPEN a
greater capacity and efficiency for the resolution of computationally expensive problems and turns it into an effective tool for integer Multiobjective Problems. The user can provide certain values such as the backtrack parameter and the cut threshold. Besides this, several options can be chosen for: selecting the next node to be examined, selecting the variable to be branched, choosing the branching option, deciding which kind of cuts can be made on the feasible set, deciding whether to use heuristic approaches or not and selecting options for the preprocessing phase. More details about the particularities of the Branch and Bound algorithm can be found in Ref. [12].

It is well known that the weights play an important role in the resolution of a Goal Programming problem. In order to avoid the possible bias effect of the solutions due to the different measurement units of the goals, MOPEN offers six different normalization options in the Goal Programming Normalizations menu. Namely, the user can choose among No Normalization, Percentage, Euclidean, $L_1$, Zero–One or Free (see Section 2.6) normalization options. More details about these normalizations and their properties can be found in Ref. [20]. For the same reason, in the generating methods, the normalization of the weights assure a better distribution of the efficient solutions obtained. Therefore, the Efficient Set Generation Normalizations menu of MOPEN offers six different options to normalize the objective functions: No Normalization, Range (Ideal–Anti-Ideal), Euclidean, $L_1$, Ideal or Free (see Section 2.6).

Finally, the Transfer Solutions to an Excel File option offers the possibility to create an Excel file containing the solution of the problem. An assistant subroutine allows the user to send the solutions corresponding to any file to an Excel spreadsheet named Excel_<Resolution Option>_.<Problem Name>.xls where the data are conveniently ordered.

2.6. Some programming aspects

In this section, some problems and challenges that have appeared during the programming process of MOPEN are mentioned, as well as the way they have been handled.

- When the Weighting Method is used as a generating technique in order to obtain efficient solutions, especially if the automatic generation option is chosen, the number of solutions calculated can be huge in not very large problems. This is why it was decided to include the filtering option of these solutions, described in Section 2.5. This way, the user can work with a more reasonable number of solutions contained in a smaller file. The same option is available when using the Constraint Method or compromise programming.

- When some zero weight is used in the weighting method, the corresponding solution is only guaranteed to be weakly efficient. This is why the efficiency test described in Section 2.4 is carried out in these cases. However, the problems ($C_j$) that are solved in this test are very likely to have a single feasible solution, $x^*$. This fact can cause instability problems when solving the problem, when it has large dimensions. This is why, in practice, a small tolerance is allowed in the constraints,

$$f_i(x)\leq f_i(x^*) + \alpha$$

and a similar tolerance is allowed for the corresponding optimal solutions.

- When using the Constraint Method as a generating technique, especially if the automatic generation option is chosen, many of the resulting constraint problems may have an empty feasible set. This fact has turned out to be the cause of computing errors when solving large problems. In order to overcome this eventuality, a Goal Programming approach has been used to test whether there are feasible solutions for the constraints of each problem. If there are feasible solutions, then the constraint problem is solved. If not, this information is transferred to the solution file, and the method proceeds to the next set of bounds.

- In general, the normalization of the weights or the functions in the different approaches are just a technical issue, in order to avoid undesired bias effects. However, the resulting solutions may be senseless for the decision maker. In a real application of our software, the decision makers expressed their wishes to divide the goals by numbers they considered significant (for example, for a function that determined the budget, a public hospital was going to receive from the government, they decided to divide it by the amount
required by the hospital). These wishes did not fit any of the existing normalizations scheme, but still they turned out to be good normalizations from both the algorithmic and the decisional points of view. Thus, the so-called free normalization option was allowed. In this scheme, the user can give a number for each objective function or goal, so that each one will be divided by the given number.

- The resolution of compromise programming problems, using metrics different from $L_1$ or $L_\infty$, cannot be carried out using linear programming schemes because the objective function is nonlinear. Currently, the authors are working on the development of a version of MOPEN for certain kinds of nonlinear problems. This will make it possible to solve more compromise programming problems.

- As is well known, a Goal Programming scheme can produce a final solution which achieves all the goals, but is not efficient for the objectives underlying each goal. In this case, the efficiency restoration schemes mentioned in Section 2.4 can be used. However, in some real applications, the decision makers expressed their wish (once the achievement of the goals was assured) to search, within the satisfying set, efficient solutions for other objective functions. This is why MOPEN allows the possibility to use different functions for the goals and for the objectives.

- When working with problems with a large number of solutions, it is very likely that the user has the data previously stored in another format (e.g., an Excel spreadsheet). In this case, the conversion of the solution files to the spreadsheet may not be an easy task for the user. This is why MOPEN offers the possibility to export the solution files to an Excel spreadsheet, so that the final solutions can be visualized in a more convenient format.

- In order to test whether MOPEN was able to solve large Goal Programming problems, many randomly generated test problems were used. Their size was up to 1000 variables, and the results were compared with those given by CPLEX. This test proved that MOPEN could handle this type of problem.

3. Example

Next, problem $(P)$ (see Section 2.3) is solved using different methods, in order to show the behavior of MOPEN. First, the Weighting Method is used to generate efficient vertices by assigning weights to the objective functions. Using a single vector or weights, the solution shown in Fig. 5 is obtained. The user can choose between visualizing the solution function values or the solution point in the decision space. Besides this, the options of changing the weights and
Solutions using the file: Sample.txt
Problem Name: Sample

<table>
<thead>
<tr>
<th>Pay-off Matrix</th>
<th>Pay-off Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^1$: $f_1(x^1) = 54$</td>
<td>$f(x^1) = 48$</td>
</tr>
<tr>
<td>$x^2$: $g_1(x^2) = -15$</td>
<td>$g(x^2) = -10$</td>
</tr>
<tr>
<td>$x^3$: $h_1(x^3) = 0$</td>
<td>$h(x^3) = 7.5$</td>
</tr>
</tbody>
</table>

$f$: ideal $= 54$ anti-ideal $= 15$
$g$: ideal $= -10$ anti-ideal $= 48$
$h$: ideal $= -7.5$ anti-ideal $= 19.5$

Weighted-Sums solutions
Normalization used: Range Normalization (ideal – anti-ideal values)

Solution Point 1:
$x = 0$
$y = 0$
$z = 0$
$t = 0$
$u = 0$
$v = 7.5$

Objective function values at this point
$f = 0$
$f$ weight $= 1 * 0.0144928$
$g = 7.5$
$g$ weight $= 1 * 0.0172414$
$h = -7.5$
$h$ weight $= 1 * 0.037037$

Fig. 6. Initial fragment of the solution file corresponding to ($P$), using the Weighting Method.

---

**Target Values**

Introduce new target values:

<table>
<thead>
<tr>
<th>Target</th>
<th>$G_f$</th>
<th>$G_g$</th>
<th>$G_h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.00</td>
<td>11.00</td>
<td>10.00</td>
<td></td>
</tr>
</tbody>
</table>

Normalization: Percentage Normalization

<table>
<thead>
<tr>
<th>Value</th>
<th>$G_f$</th>
<th>$G_g$</th>
<th>$G_h$</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.00</td>
<td>11.00</td>
<td>9.50</td>
<td></td>
</tr>
</tbody>
</table>

Relax Target Values

Relax all the unachieved target values within a 0.100%
Relaxing Parameter: 0.1

Fig. 7. Resolution of ($P$) by Goal Programming.
changing the normalization scheme are offered at this stage. This way, different solutions can be obtained using this method. Fig. 6 shows a fragment of the corresponding solution file (Sol_WeightedSums_example.txt).

It can be observed that the weights used have been factorized so that both the original weight given by the user and the normalization factor can be seen. For example, for function $f$, the original weight is 1, and the normalizing factor is 0.0144928. This way, the user can visualize the solutions for the original weights, because the normalized ones can be meaningless from a preferential point of view. On the other hand, an experienced analyst can find out whether the normalizing option chosen is effective enough or not. If some zero weight is given, then the efficiency test described in Section 2.4 is carried out, and this information is transferred to the solution file.

Next, problem $(P)$ will be solved using the Lexicographic Goal Programming scheme. The Percentage option has been chosen to normalize weights assigned to the goals, and the Interactive option has been used to restore the efficiency of the original solution. In the example, there exist solutions that satisfy all the goals for the given target values, as shown in Fig. 7. This screen lets the user analyze the current satisfying solution. It is possible to change one or some target values, giving new absolute values for them or to relax all the unachieved target values up to a given percentage. If the Proceed to Restoration option is chosen, MOPEN starts the interactive restoration algorithm. In this case, the screen shown in Fig. 8 appears, where the current state of each goal (efficient, dominated or unbounded, see Ref. [20]) is displayed.

In our example, it can be seen that all goals are dominated. Goal $G_f$ was chosen for restoration, and an efficient solution was obtained. All the results of the Goal Programming process, including the restoration scheme, are transferred to the file Sol_GoalProgrm_Sample, which is shown in Fig. 9.

The information regarding the pay-off matrix is shown at the beginning of the file. It must be pointed out that a goal is not, and does not necessarily come from, a criterion to be maximized or minimized. Thus, it may seem senseless to talk about the corresponding pay-off matrix or the ideal or anti-ideal values of a goal. Nevertheless, as this information may be helpful for the user in order to give the target values, it is assumed that a goal $f_i(x) \leq u_i$ is identified with a criterion to be minimized, and a goal of the form $f_i(x) \geq u_i$ is identified with a function to be maximized. This is the way the elements of the pay-off matrix are obtained. Therefore, the user must be aware of this identification when interpreting the values of the pay-off matrix. For example, for equality goals or goals with a nonclearly defined optimization (maximization or minimization) character, the pay-off matrix may not give significant information.

Next, the solutions of each priority level are shown. That is, the solution point, the values of its corresponding deviation variables, the optimal value

![Interaction Window](image-url)
Fig. 9. Lexicographic Goal Programming solution file for problem \((P)\).
of the achievement function and the current values of all the goals are displayed. If the optimal solution obtained for the current priority level is not unique, this fact is also pointed out by MOPEN. This information is very helpful to the user, in order to identify possible redundant levels, and to evaluate how optimistic or pessimistic the target values are.

Finally, the information regarding the efficiency restoration procedure is shown including the status (efficient, dominated or unbounded) of each goal in each iteration and finishing with the final satisfying and efficient solution of the problem.
4. Conclusions

The computational package MOPEN has been described in this paper. MOPEN has been designed in order to use the main Multiobjective Programming and Goal Programming techniques, for linear problems with continuous and/or integer variables. Besides this, this package has been implemented under Windows environment. Thus, the user can enjoy all the advantages of such an implementation, i.e., a friendly interface, simple and intuitive data entry and layout procedures, etc. Generally speaking, the package has been designed in order to facilitate the resolution of linear multiple criteria problems of any kind (continuous, integer or binary variables), with a wide variety of methods (Efficient Set Generation, Goal Programming, Compromise Programming or Reference Point). The programs make it easy to build the data entry file, edit any of the files related to the solution of the problem, generate the weights for the criteria, edit the solution in a Excel spreadsheet, visualize the problem data and, fundamentally, apply any of the previously mentioned methodologies. These facts, together with a reasonably good performance, as shown in Section 2.1, turn MOPEN into a powerful tool for solving Linear Multiobjective problems, and it can be easily used in both teaching or research environments.

The wide variety of options in all the methods (normalization, determination of weights, efficiency restoration, etc.) can be very useful for an experimented analyst. In this sense, it must be pointed out that MOPEN has not been designed in order to substitute the analyst. On the contrary, an experienced analyst is necessary in order to use this package and make the best use of its potential capabilities.

Finally, there are two main future research lines in this field. First, this implementation must be validated, using it to solve real Multiobjective problems. Besides this, the Meta-Goal Programming option (see Ref. [14]) will be included. Second, a version of the package for nonlinear problems is presently being designed.

Acknowledgements

The authors wish to express their gratitude to the anonymous referees who have made significant contributions to improve the quality of the original manuscript. This research has been partially founded by the research projects SEJ 417 (Andalusian Regional Government) and SEC 2001-1742, BFM 2001-1844 and BFM 2002-11282-E (Spanish Ministry of Science and Technology).

References


**Rafael Caballero** is a Professor in the Department of Applied Economics (Mathematics), University of Málaga, Spain. He holds a PhD degree in Mathematics obtained from the University of Málaga. He is interested in the field of Multiple Objective Programming (linear, quadratic, convex, combinatorial, etc.). Presently, his research is based on heuristics methods and application to problems in the public sector.

**Francisco Ruiz** is an Assistant Professor in the Department of Applied Economics (Mathematics), University of Málaga, Spain. He holds a PhD degree in Economics Sciences obtained from the University of Málaga. His research has been carried out in the field of linear, quadratic, convex, combinatorial, hierarchical Multiple Objective Programming. Presently, he is interested in interactive methods.

**Mariano Luque** holds a PhD degree in Economics Sciences and holds a senior lecturing post at the University of Málaga. Nearly all his research has been carried out in the field of Hierarchical Multiple Objective Programming. Presently, his research is based on interactive methods.

**Julian Molina** holds a PhD degree in Economic Science obtained from the University of Málaga. His research has been carried out in the field of Multiple Objective integer and continuous Programming. Presently, his research is based on metaheuristic algorithms for Multiple Objective Programming problems and their applications.