Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 1, 2004

Multiobjective fuzzy optimization method

Gabriel Oltean[1]

Abstract -The paper proposes a new multiobjective optimization method, based on fuzzy techniques. The method performs a real multiobjective optimization, every parameter modification taking into account the unfulfillment degrees of all the requirements. It uses fuzzy sets to define fuzzy objectives and fuzzy systems to compute new parameter values. The strategy to compute new parameter values uses local gradient information and encapsulates human expert thinking. After introducing our optimization method, we optimize the design of a finite response filter.

Keywords: multiobjective, fuzzy objective, fuzzy system, population of solutions

I. INTRODUCTION

Mathematical formulation of general optimization problem (GP - General Problem) is [1],[2]:

$$\begin{aligned} &\text{Find } x \text{ that} \quad \text{minimize} \quad f_0(x) \\ &\text{subject to:} \quad g_j(x) \le 0, \quad j = 1,...,m \\ &h_q(x) = 0, \quad q = 1,...,p \\ &x_l \le x \le x_u \end{aligned} \tag{1}$$

It is formulated an constrained optimization problem with a single objectiv (a scalar one). The rigidity of the mathematical problem posed by the general optimization formulation is often remote from that of a practical design problem. Rarely does a single objective with several hard constraints adequately represent the problem being faced. More often there is a vector of objectives $f(x) = \{f_1(x), f_2(x), \ldots, f_o(x)\}$ that must be traded off in some way. Multiobjective optimization is concerned with the minimization of a vector of objectives $f(x)$ that may be subject of a number of constrains or bounds:

$$\begin{aligned} &\text{Find } x \text{ that} \quad \text{minimise} \quad \{f_1(x), f_2(x), \ldots, f_o(x)\} \\ &\text{subject to:} \quad g_j(x) \le 0, \quad j = 1,...,m \\ &h_q(x) = 0, \quad q = 1,...,p \\ &x_l \le x \le x_u \end{aligned} \tag{2}$$

Because $f(x)$ is a vector, if any of its components are competing, there is no unique solution to this problem. Instead, the concept of noninferiority (also called Pareto optimality) must be used to characterize the objectives [2], [1], [4]. A noninferior solution is one in which an improvement in one objective requires a degradation of another. The techniques for multiobjective optimization are wide and varied. Goal attainment method of Gembicki involves expresing a set of design goals $f^* = \{f^*_1, f^*_2, \ldots, f^*_o\}$ associated with the set of design objectives. The multiobjective optimization problem is then expressed as a standard optimization problem using the formulation [5]:

$$\begin{aligned} &\text{minimise } \gamma, \quad \gamma \in R \\ &\text{such that } f_k(x) - w_k(x)\gamma \le f_k^r; \quad k = 1,...,o \end{aligned} \tag{3}$$

In order to transpose the real word problem in the mathematical language for the optimization first the objective functions should be defined. The objective functions are the measures between the actual values of the functions and the required values. The objective functions must be carefully selected so that they lead to the requirements achievement.

In this paper, we propose a new fuzzy multiobjective optimization method. Our method simultaneously optimizes all the objectives so it directly solves the multiobjective problem, without any transformation into an one objective problem. The algorithm relays on fuzzy sets to formulate the optimization objectives and on fuzzy systems to compute the new values for the variables in every optimization iteration. Because it is a gradient method, a population of solution can be employed in order to dramatically increase the chance to obtain the best possible solution.

II. THE FUZZY OPTIMIZATION METHOD

The optimization method should converge to a global optimal solution in a reduced number of iterations. This is not a simple task due to the complex relations between variables and nonlinear multi-variables functions to be optimized. A variable can affects quite different more than one function at a time, so when it is modified in order to improve one objective function it can damage another.

A. *Formulation of the optimization problem*

Consecrated formulation of the optimization problem is somehow rigid and does not always reflect the reality. Such a formulation, very restrictive, reduces the possibility to make trade-offs, a very important

[1] Technical University of Cluj-Napoca

factor in the optimization. Consequently, the solution space is confined and in many cases, an optimal solution does not exist. One way to overcome these drawbacks is to use fuzzy sets to define optimization objectives. We will fuzzify the requirements getting this way the possibility to consider different degrees for requirement achievements and acceptability degrees for a particular solution. We will associate with each requirement one or two fuzzy sets whose membership functions will represent the corresponding fuzzy objective functions. For example for the requirements "greater or equal" $f_k(x) \geq f'_k$, and "equal" $f_k(x)=f'_k$ the corresponding fuzzy objective functions are presented in Fig 1.

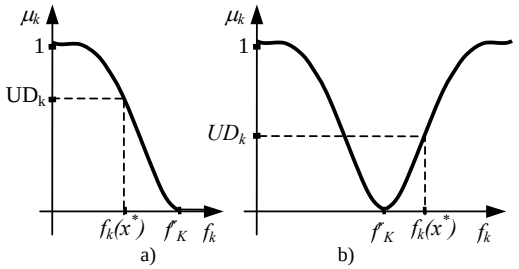

Fig.1. Fuzzy objective functions:
a) $f_K \geq f'_K$; b) $f_K = f'_K$

The fuzzy objective functions are

$$\mu_k(f_k(x)) : \quad D_{f_k} \rightarrow [0,1] \qquad (4)$$

where D_{f_k} is the range of possible values for $f_k(x)$. $\mu_k(f_k(x))$ indicates the error degree in accomplishing the i[th] requirement, so we will call them unfulfillment degrees (*UD*). A value μ_k=0 means a fully achievement of the fuzzy objective, while a value μ_k=1 means that fuzzy objective is not achieve at all, this occurs when $f_k(x)$ takes an unacceptable value. In Fig.1. We can see, for the current value of the variables vector x^* the corresponding value of the unfulfillment degree is UD^*. Our new multiobjective optimization problem formulation is:

Find *x* that
minimise $\{\mu_1(f_1(x)), \mu_2(f_2(x)), ..., \mu_o(f_0(x))\}$ (5)

B. *The idea of population of solutions*

Starting the optimization with only one initial solution, we can remain blocked into a local Pareto optimal point, where an improvement in one objective requires a degradation of another. If we can obtain a set of local Pareto optimal points, it is highly possible to have the global Pareto optimal point among them.
So, instead of using one search path we suggest using a parallel search dealing with the idea of population of solutions consisting of candidate solutions. The optimization starts with the initial candidate solutions. In our implementation, these initial candidate solutions can be obtained in several ways: randomly generated, generated with Latin Hypercube Technique, or user provided.
In each iteration, for every candidate solution the actual function value, the UDs and new parameter values are computed. If the UDs for one candidate can not be decreased anymore, we have found a local Pareto optimal point and the future iterations will not visit this candidate solution, shortening the entire optimization time.
The optimization algorithm stops in one of the following situations:
i) all the UDs become zero for one candidate solution. This candidate solution is considered a global Pareto optimal point and it is our final solution. We will not continue to search other Pareto optimal point on the remaining search paths.
ii) none of the candidate solutions can be further improved, meaning that the set of local Pareto optimal points was obtained. As the final optimal solution we chose the one with the minimum value of the mean of unfulfillment degrees (MUD), considered as global optimal point.
Also the algorithm will stop if the maximum number of iterations is reached.

C. New parameter values computing

The method for computing the new values for the variables involves fuzzy techniques and local gradient information.
Each variable can affect more or less each objective function. In our method the sign and the value to modify a certain variable takes into account the UDs, the gradients and the relative importance of the involved variables in relations with the objective functions.
Our method acts as a human expert for a certain circuit performance:

- it is better to modify more the parameter with greater importance, because it can really affect the performance, and the modification also depends on the unfulfillment degrees of the corresponding requirements.
- the parameter with lower importance is modified less or not at all, because its influence on circuit performance is insignificant.
- the final modification of a parameter is a weighted sum of the partial modification (imposed by every objective function).

Such human expert knowledge is captured and incorporated in our method by means of a fuzzy logic system. The algorithm to compute the new variable values follows:

In each iteration:

i) Compute the local gradients of the functions in relation with each variable, $\nabla f_k(x_i)$ - the local gradient of f_k function in relation with x_i variable.
ii) For every function f_k we compute the importance of the variables $v_{f_k}(x_i)$ that shows the relative importance of every x_i in modifying the function f_k.

These importance of the variables are computed based on absolute values of the local gradients:

$$v_{f_k}(x_i) = \frac{|\nabla f_k(x_i)|}{\sum_{i=1}^{n} |\nabla f_k(x_i)|}; i = 1,...n; k = 1,..., o \quad (6)$$

iii) For every requirements $f^r{}_k$ compute the UD_k as a membership degree of the actual value of the corresponding function $f_k(x^*)$ to the associated fuzzy objectives. Two examples are shown in Fig.1.

iv) For every variable x_i and every function f_k we compute a partial coefficient to modify that parameter. This partial coefficient $coef_{xi}(f_k)$ is computed by a first order Takagi-Sugeno fuzzy system (Fig. 2.)

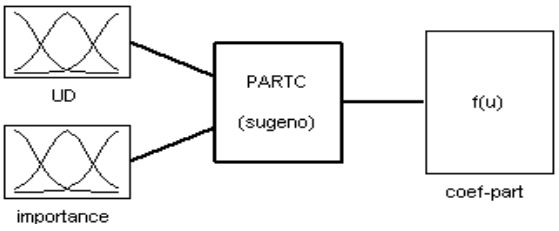

Fig.2. Partial coefficient computing

The fuzzy sets for the input linguistic variables "UD" and "importance" and for output linguistic variable "coef-part" are not presented here due to the lack of space.

The fuzzy rules are presented in Table 1. where, for example the 4th column and the 3nd row give the following fuzzy rules:

"If UDR is Medium and importance is Small then coef-part is Small".

Table 1

UD / Importance	Z	S	M	L
Z	**Z**			
S		**VS**	**S**	M
M		**S**	**M**	L
L		S	L	VL

Z – Zero
VS – Very Small
S -Small
M – Medium
L – Large
VL – Very Large

The control surface generated by this fuzzy system is presented in Fig. 3

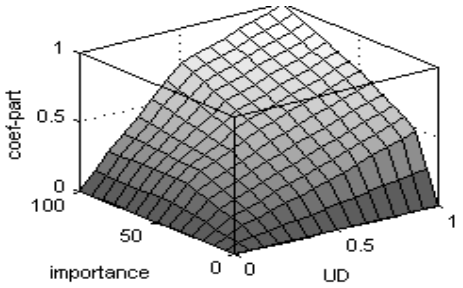

Fig.3. Control surface to compute partial coefficient

v) The partial coefficients $coef_{x_i}(f_k)$ receive a plus or minus sign depending on the sign of the local gradient and on the direction (go up or go down) in which the function must be modified. So we obtained partial coefficients with sign: $scoef_{x_i}(f_k)$.

vi) For every variable x_i we compute the function influence upon variable modification $p_{x_i}(f_k)$ that shows the relative importance of every function f_k to compute the modification of x_i parameter.

$$p_{x_i}(f_k) = \frac{|\nabla f_k(x_i)|}{\sum_{k=1}^{o} |\nabla f_k(x_i)|}; k = 1,..., o; i = 1,..., n \quad (7)$$

vii)The coefficients used for modifying each parameter are computed as weighted sum of the partial coefficients, the weight being the influences $p_{x_i}(f_k)$. It means that the greater the influence is, the greater the contribution on the partial coefficient.

$$scoef_{x_i} = \frac{\sum_{k=1}^{o} \left(p_{x_i}(f_k) \cdot scoef_{x_i}(f_k)\right)}{\sum_{k=1}^{o} \left|p_{x_i}(f_k)\right|} \quad (8)$$

viii) Compute new variable values:

$$x_i^{new} = x_i + scoef_{x_i} \cdot abs(x'_i) + xmin \quad (9)$$

where x'_I takes the value of x_i from 3 iterations back if in all these 3 iterations we have the same modification sign for it as in the actual iteration. Otherwise x'_I takes the value of x_i from the current iteration. We found that using x'_iinstead of simple x_i we can change the variable sign and improve the convergence of the algorithm. Also more help in the sign changing, convergence and accuracy of final solution can by obtained using the variable *xmin*. It takes a default value to the beginning of the optimization, and that value is dynamically decreased if some oscillations appear in the mean of unfulfillment degree.

Finally, we should mention that the optimization method acts in an adaptive manner: when the UDs are large (towards 1) we have large coefficients to modify the variables (see Table 1). For small UDs we have small coefficients to modify the variables, so we can focus our search so that the solution converges to the exact local Pareto optimal point.

III. IMPLEMENTATION

In order to check and validate our multiobjective optimization algorithm we implemented a prototype system in Matlab for Windows. The prototype consists on a main function "optfuzz" and other secondary function. The main function should be

invoked from Matlab workspace with a series of arguments :

- fun – a string containing the name of the Matlab function that computes the objective functions;
- reqs – vector of numerical values of the requirements;
- sign – vector with +1, –1 or 0 values, with the same length as reqs vector. When the values is +1 "optfuzz" attempt to make the objective function greater or equal to corresponding requirements; for –1 "optfuzz" attempt to make the objective function less than the corresponding requirements; for 0"optfuzz" attempt to make the objective equal with the corresponding requirements
- nrvar – number of variables
- weight – vector with weight for the objective functions
- proc – vector with values in (0,1) that control the fuzzy sets defining fuzzy objectives
- lb – a vector of lower bounds of the variables;
- ub – a vector of upper bounds of the variables;
- init_sol – variable that set the method for generating the initial solution
- options – vector with some options of the optimization algorithm (number of iterations, number of candidate solutions, initial value for *xmin*)

The user can provide empty values for some of the above arguments; in this case, the default values are used.

The user should only write his objective functions and run the "optfuzz" with the arguments show above. The optimization routine return the final values of objective functions, the values of the variables, the UD for each requirements and a curve with the evolution of MUD during the optimization for the candidate solution that provide final solution.

IV. RESULTS

In order to highlight the behavior of our new fuzzy multiobjective optimization method, we use it to solve some multiobjective optimization problems.

Consider designing a linear-phase Finite Impulse Response (FIR) filter. The problem is to design a low pass filter with magnitude one at all frequency between 0 and 1.0 Hz and magnitude zero between 0.15 and 0.5 Hz. The frequency response $H(f)$ for such a filter is defined by

$$H(f) = \sum_{n=0}^{2M} h(n) e^{-j2\pi fn} = A(f) e^{-j2\pi fM}$$
$$A(f) = \sum_{n=0}^{M-1} a(n) \cos(2\pi fn) \quad (10)$$

where *A(f)* is the magnitude of the frequency response.

So the problem is to compute the magnitude coefficients *a(n)* so that the magnitude response matches the desired response (at each frequency) with some tolerance. We must use the discretization of the frequency domain we are interested in. The number of function to be optimized equals the number of discrete frequency, and the number of variables equals the number of *a* coefficients.

First we use only 5 (uniform distributed) frequency in each domain, so a number of 10 function to optimize and a number of 15 variables. The value of the requirements are 1 for the five frequency in [0; 0.1] range, and 0 for the five frequency in [0.15; 0.5]. Because frequency between 0.1Hz and 0.15 Hz are not specified, no requirements are needed here.

We run the optimization algorithm for a population of 9 candidate solutions, for 150 maximum number of iterations, with randomly generated initial solutions. In order to see how the internal computations deploy, we reproduced in Fig.4. the evolution of 3 (from a total of 10) UDs during the optimization, for 1st candidate solution.

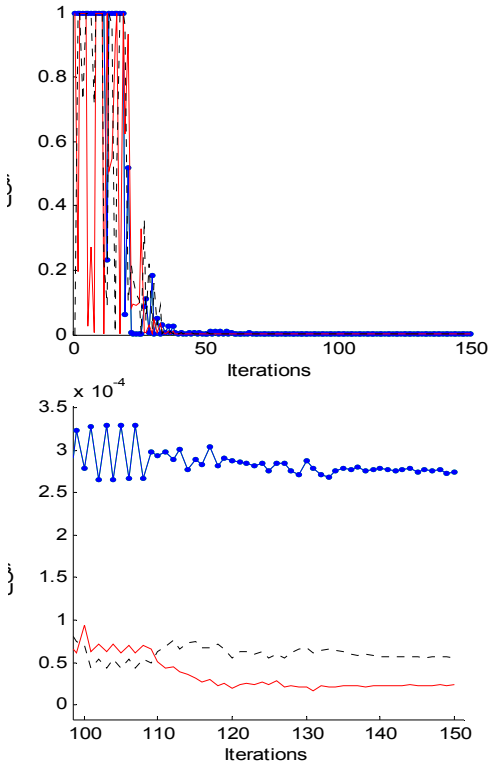

Fig. 4. Evolution of there UDs.
a) Full process; b) Details: final iterations

From the Fig.4. a) one can see that in the first iterations (up to around 40) all the UDs have (large) variations. This is because we are far from a good solution and each function asks for high modification of the variables. Remember that each function depends on each variable. After this "transient" regime, all UDs falls towards zero and continue to decrease, up to an magnitude order of 10^{-4} in the final iterations (fig.4. b)), to reach, as close as possible, a 0 value for UDs. Fig.5. depicts the evolution of the mean unfulfillment degree MUD (arithmetic mean of all 10 UDs), that globally characterize the

optimization process, also for the 1st candidate solution. In the first 50 iterations, large oscillations (due to large changes in each UD) and then a rapid improvement in the value of MUD can be seen. The algorithm is very close to a good solution (MUD=0.00676355294315, in iteration 50) After that, the algorithm try to improve the solution, continuing

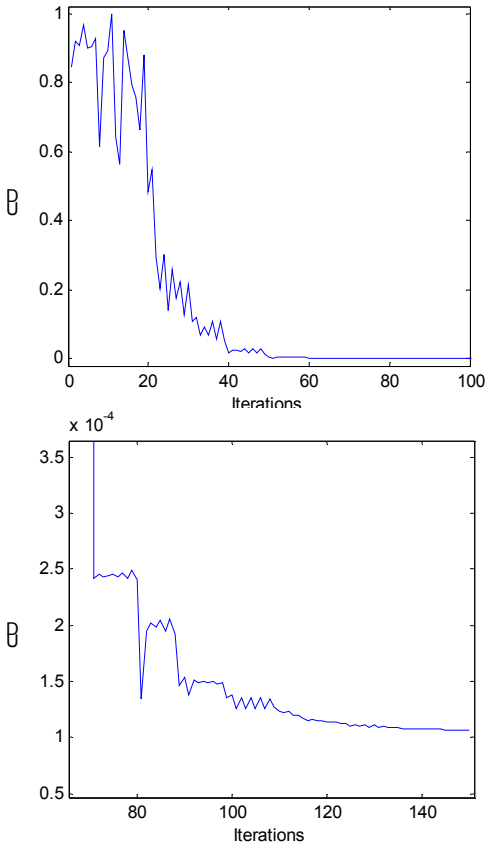

Fig. 5. Evolution of the mean UD.
up: full process; bottom: details for final iterations

to decrease the MUD up to the 0.00010635653417 value in iteration 150. The convergence is slower, because we are close to the ideal solution, so the algorithm should "move carefully around". If one will run the algorithm for a larger number of iteration a better solution can be reach. Let's mention that the necessary time to run the optimization (9 candidate solution, 10 functions, 15 variable, 150 iteration) was 583s on a Pentium IV, 1GHz, 256 Mo RAM machine.
The results obtained for all 9 candidate solution are presented in Table 2 and Table 3. Table 2 contains information about the initial MUD (for initial value of the variables) and final MUD (for value of variables after optimization). For all candidate solutions, there is a very good evolution of unfulfillment degrees from approximately 1 to 0.01 or even less. So, indeed, this multiobjective problem has more than one good solution. The best solution is given by the 1st candidate solution with a final value of MUD almost zero, an acceptable solution for a practical problem.

Table 2

CANDIDATE SOLUTION	MUD	
	initial	final
1°	0.843818	0.000106
2°	1.00	0.015298
3°	0.950626	0.010040
4°	1.00	0.049173
5°	0.900934	0.004372
6°	1.00	0.034177
7°	1.00	0.004431
8°	1.00	0.001444
9°	0.943818	0.000176

Table 3 present the initial and final values for every function to be optimized together with the corresponding requirements, for three candidate solutions (1°, 4° and 5°). We can see that even for the solution 4° that is the poorest from the initial set, the final values of the function are very close to the requirements (e.g. 0.989 for 1; 1.006 for 1; 0.006 for 0, -0.002 for 0). For the best solution (1°) the differences are smaller (e.g. 1.0009 for 1; 0.9992 for 1; -0.00001 for 0 and so on).

Table 3

REQ	CAND. SOL. 1^0		CAND. SOL. 4^0		CAND. SOL. 5^0	
	init.	final	init.	final	init.	final
1	15.00	0.99834	3.97	0.96669	-2.15	0.98948
1	5.34	1.00181	0.01	1.03974	-0.11	1.01156
1	-2.65	0.99824	-2.26	0.96304	1.65	0.98880
1	1.61	1.00099	2.65	1.02203	-0.28	1.00651
1	0.99	0.99921	5.01	0.98105	-3.31	0.99505
0	1.48	0.00092	4.07	0.01999	-0.41	0.00612
0	075	-0.00051	3.52	-0.01199	1.81	-0.00294
0	-0.07	-0.00001	-0.50	-0.00403	-1.60	-0.00218
0	0.43	0.00022	6.12	0.00518	-0.01	0.00097
0	1.0	0.00021	5.74	0.00693	-5.54	0.00180

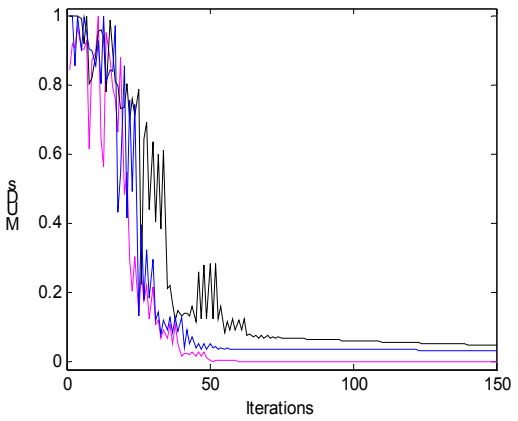

Fig. 6. Evolution of MUDs for the candidate solutions 1°, 4° , 5°

Also the evolution of the MUDs for these three candidate solutions are presented in Fig. 6.
Now, let us consider a more complex situation. For the same FIR application take 50 (uniform distributed) frequency in each domain, so a number of

100 function to optimize and a number of 15 variables. The problem is two-fold complicated. First, the number of function to be optimized is higher (100 instead of 10). Second, we have only 15 variables to set the required value for each function. After running the optimization we reached our best final solution after 57 iteration, with a final MUD of 0.013577363182378. The evolution of the MUD during the optimization is presented in Fig. 7. Further iterations can not improve the solution.

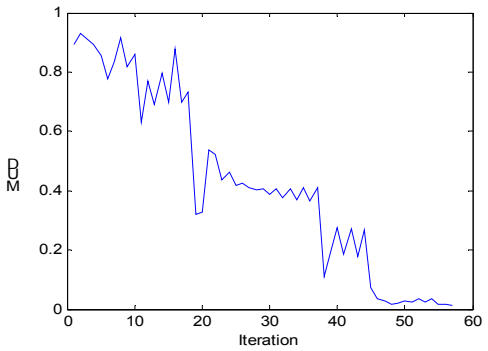

Fig.7. Evolution of MUD for 100 optimization functions

To see the power of our method we compared the previous result with the result obtain with the Goal Attainment method. The Goal Attainment method is also a multiobjective optimization method, implemented in the Optimization Toolbox from Matlab [5]. For the same problem, with the same initial point (1 for all 15 initial variables), the results obtained with both method are presented in Tabel 4. In order to have the same measure for both method we computed the absolute error between the function value after optimization and the required value, for every function. The mean absolute error over all 100 functions shows that our method (Fuzzy multiobjective optimization) provided a more accurate solution than the Goal attainment method, 0.009502 being les than 0.017347. The price paid is a larger number of iterations and accordingly more time to complete the optimization. Anyway, the time for our method remains small enough for practical applications.

Table 4

METHOD	MEAN ABS. ERROR	MAX. ABS. ERROR	ITER.	TIME
Fuzzy	0.009502	0.055248	57	250s
Goal attain.	0.017347	0.025996	4	4s

Fig. 8. presents a comparison of the magnitude response computed with the variables (the *a* coefficients) provided by the optimizations method (up – fuzzy multiobjective optimization and bottom – Goal attainment optimization) with the ideal magnitude response. We can easily see that both optimization methods ensure nice frequency characteristics. The characteristic provided by our optimization method (up) is closer to the ideal one, having smaller oscillations than the other one (bottom).

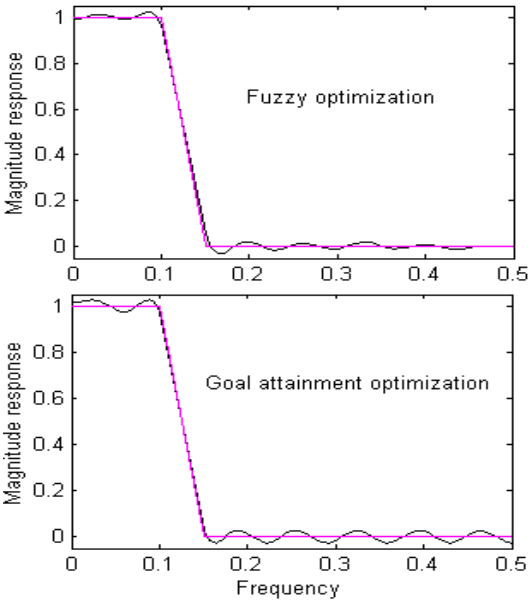

Fig. 8. Magnitude response with variable values provided by Multiobjective optimization – up; Goal Attainment - bottom

V. CONCLUSION

In this paper a new multiobjective optimization method using fuzzy logic has been introduced. The method really allows optimization of several objectives simultaneously because the modification of each parameters is a function of the unfulfillment degrees of all the requirements.

The results obtained after optimizing the coefficients of a FIR filter show that our method works very well. Due to the population of solutions, we can find a set of optimal points. The method has a very large chance to find the global optimal solution due to its multiple search paths. Also in the proximity of the final solutions, the method works well to continue decrease MUD up to the local optimal points. The quality of each final solution is very high. This is possible because the method uses local gradient information and works in an adaptive manner: while the UDs decrease, the step in the parameter modification also decreases. Compared with other multiobjective optimization method (Goal Attainment) our method assures a better accuracy of the final solutions.

REFERENCES

[1] Boyd, S., Vandeberghe, L., Introduction to Convex Optimization with Engineering Application, Stanford University, 1999.

[2] Branch, Mary Ann, Grace, A., Optimization Toolbox. For Use with Matlab, The MathWork Inc., 1996

[3] Grimbleby, J., B., Computer-Aided Analysis and Design of Electronic Networks, Pitman Publishing 1990, pp.157-190;

[4] Maulik, P.C, Comments on "FPAD: A Fuzzy Nonlinear Programming Approach to Analog Circuit Design", IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, No.6, June 1997, pp.656;

[5] ****, Optimization Toolbox Help for Matlab R13, The MathWork Inc. 2002.