GDF v2.0, an enhanced version of GDF

Ioannis G. Tsoulos [a,1], Dimitris Gavrilis [b], Evangelos Dermatas [b]

[a] *Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina 45110, Greece*

[b] *Department of Electrical and Computer Engineering, University of Patras, Patras 26500, Greece*

Abstract

An improved version of the function estimation program GDF is presented. The main enhancements of the new version include: multi-output function estimation, capability of defining custom functions in the grammar and selection of the error function. The new version has been evaluated on a series of classification and regression datasets, that are widely used for the evaluation of such methods. It is compared to two known neural networks and outperforms them in 5 (out of 10) datasets.

PACS:02.30.Mv, 02.60.-x, 02.60.Ed, 07.05.Mh.

Key words: Function approximation, stochastic methods, genetic programming, grammatical evolution.

NEW VERSION PROGRAM SUMMARY

Manuscript Title: GDF v2.0
Authors: Ioannis G. Tsoulos, Dimitris Gavrilis, Evangelos Dermatas
Program Title: GDF v2.0, an enhanced version of GDF
Journal Reference:
Catalogue identifier:
Licensing provisions: none
Programming language: GNU C++
Computer: The program is designed to be portable in all systems running the GNU C++ compiler
Operating system: Linux, Solaris, FreeBSD
RAM: 200000 bytes
Number of processors used: 1
Keywords: Function approximation, stochastic methods, genetic programming, grammatical evolution.

[1] Corresponding author

PACS: 02.30.Mv, 02.60.-x, 02.60.Ed, 07.05.Mh.
Classification: 4.9
Catalogue identifier of previous version: ADXC_v1_0
Journal reference of previous version: Comput. Phys. Comm. 174(2006)555
Does the new version supersede the previous version?: Yes
Nature of problem:
The technique of function estimation tries to discover from a series of input data a functional form that best describes them. This can be performed with the use of parametric models, whose parameters can adapt according to the input data.
Solution method: Functional forms are being created by genetic programming which are approximations for the symbolic regression problem.
Reasons for the new version: The GDF package was extended in order to be more flexible and user customizable than the old package. The user can extend the package by defining his own error functions and he can extend the grammar of the package by adding new functions to the function repertoire. Also, the new version can perform function estimation of multi - output functions and it can be used for classification problems.
Summary of revisions: The following features have been added to the package GDF:

(1) Multi - output function approximation. The package can now approximate any function $f : R^N \to R^M$. This feature gives also to the package the capability of performing classification and not only regression.

(2) User defined function can be added to the repertoire of the grammar, extending the regression capabilities of the package. This feature is limited to 3 functions, but easily this number can be increased.

(3) Capability of selecting the error function. The package offers now to the user apart from the mean square error other error functions such as: mean absolute square error, maximum square error. Also, user defined error functions can be added to the set of error functions.

(4) More verbose output. The main program displays more information to the user as well as the default values for the parameters. Also, the package gives to the user the capability to define an output file, where the output of the gdf program for the testing set will be stored after the termination of the process.

Running time: Depending on the train data.
Other comments: A technical report describing the revisions, experiments and test runs is packaged with the source code.

LONG WRITE-UP

1 Introduction

This technical report explains in detail the installation and the using of the function estimation program named GDF with the revisions that have been made in version 2 of the package. The package utilizes the Grammatical Evolution procedure[1] in order to create functional forms which can approximate any function $f : R^N \to R^M$. The problem of function estimation may be defined as follows: Given K points and their associated values (x_i, y_i), $x_i \in R^N$, $y_i \in R^M$ estimate a function $f : R^N \to R^M$ that minimizes some error function between the real values y_i and the estimated values $f(x_i)$. The multi-output function approximation feature, which has been added in the GDF2 version of the packages, allows the program to be also used in classification problems (apart from regression). This feature resembles multi - objective optimization [2–4], a simultaneous optimization of problems with multiple objectives (which under single - optimization their solutions conflict with each other). In the current implementation, each objective can be encoded in a separate function to be estimated (optimized). Since the optimal solution is represented by the chromosome, all the objectives participate in the fitness of the chromosome. Furthermore, with the introduction of custom defined error function, weights can be introduced in each objective function to be optimized and the weighted sum $f(x) = \sum_{i=1}^{m} w_i F_i(x)$ can be defined as the error function, which is known as fitness sharing.

The rest of this report is organized as follows: in section 2 of this report the contents of the distribution and the installation steps are presented, in section 3 the command line parameters of the program *gdf* are described along with the command line enhancements of the revised version of the package, in section 4 the results from the application of the program to different regression and classification problems are listed and compared to two other methods for regression and classification and finally in section 5 some examples from test runs are listed.

2 Installation procedure

The package is distributed in a tar.gz file named GDF2.tar.gz and under most UNIX systems the following commands must be issued in order to to extract and install the package:

(1) gunzip GDF2.tar.gz

(2) tar xfv GD2.tar
(3) cd GDF2
(4) make

The first two steps create a directory named *GDF2* and the *make* command will compile the package and it will add the main executable *gdf* as well as the grammar files under the *bin* subdirectory. The compilation is performed with the use of GNU C++ compiler, but the user can choose a different compiler by editing the file Makefile under the *GDF2* directory and changing the line *CC=c++* to the desired compiler, e.g. *CC=pgCC*.

3 The program gdf

The program *gdf* is the main executable of the package and it has a series command line parameters. In the revised version of the *GDF* package three additional parameters have been added to the parameter set and the executable supplies more information to the user for the default values of the parameter when he requests it. The program takes the following parameters in the command line:

(1) **-h**: The program prints a help screen with all the command line parameters and their default values and it terminates.
(2) **-g** grammar_file: The string parameter grammar_file specifies the grammar specification in BNF notation, that will be used by the program. The default value for this parameter is *grammar.txt*. The revised version of the package supports user defined functions (UDF), which can be included to the optimization process by defining as grammar_file the file *udf_grammar.txt* under the subdirectory *bin*. Also, the user must edit the file *udf.cc* under subdirectory *src* and the functions udf1, udf2 and udf3 must change according to the needs of the user. Afterwards, the package must be recompiled. With this addition the new GDF tool can become a very powerful classifier or used for neural network construction and training. A full example using User Defined Functions is shown in section 5.
(3) **-p** problem_file: The string parameter problem_file determines the files containing the points where the data fitting procedure will be applied. The file must be formatted according to the scheme of figure 3. The integer number N in the file determines the dimensionality of the objective problem and the integer number K determines the amount of points in the file. Every line determines a point where the data fitting procedure will be applied. The dimensionality of the output (number M) it is discovered automatically by the program and hence the user does not need to specify this number.

Fig. 1. The format of input files

N
K

$$\begin{array}{cccccc} x_{11} & x_{12} & \dots & x_{1N} & y_{11} \dots y_{1M} \\ x_{21} & x_{22} & \dots & x_{2N} & y_{21} \dots y_{2M} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{K1} & x_{K2} & \dots & x_{KD} & y_{K1} \dots y_{2K} \end{array}$$

(4) **-t** test file: The string parameter test_file determines a file in the same format as the problem_file, where the produced functions will be applied after the termination of the process.
(5) **-c** count: The integer parameter count specifies the number of chromosomes in the genetic population. The default value is 500.
(6) **-l** length: The integer parameter length specifies the length of each chromosome in the genetic population. The default value is 100.
(7) **-s** rate: The float parameter rate defines the selection rate for the genetic algorithm. The default value for this parameter is 0.1 (10%).
(8) **-m** rate: The float parameter rate defines the mutation rate used in the genetic algorithm. The default value for this parameter is 0.05 (5%).
(9) **-n** generations: The integer parameter generations specifies the maximum number of allowed generations for the genetic algorithm. The default value for this parameter is 500.
(10) **-r** seed: The integer parameter seed specifies the seed for the random number generator and the default value is 1.
(11) **-o** file:This is an option added in the revised version of the software. The string parameter file defines the file, where the output of the *gdf* program for the testing set will be stored after the termination of the program. The default value for this parameter is the empty file, which means that we do not want any output to be stored. The file is formated in a way which can be used by other programs such as gnuplot. The format of the output file has as follows:
    (a) The first line is the dimension of the function to be estimated (M).
    (b) The second line is the number of patterns in the test set.
    (c) In every one successive line there are M columns with the values of the original function and M columns with the values of the function estimated by the *gdf* program.
(12) **-e** function. This is a new parameter added in the revised version of the software. The integer parameter function specifies the desired error function for the regression procedure. All the error functions are specified in the *errofunction.cc* file in the distribution. The available values for the parameter are:
    (a) **1** (Mean Squared Error). This is the default value for the parameter. The MSE Function was the only error function supported in the first version of the distribution. The value for Mean Squared Error is

calculated by the formula:

$$E_{\mathrm{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}(y_{ij}-t_{ij})^2 \tag{1}$$

where N is the total number of input patterns, M is the dimension of the function to be estimated, y_{ij} is the output of the function constructed by the grammatical evolution procedure and t_{ij} is the output of the function to be estimated.

(b) **2** (Mean absolute square error), which is given by the formula:

$$E_{\mathrm{ABS}} = \frac{1}{N}\sqrt{\sum_{j=1}^{M}(y_{ij}-t_{ij})^2} \tag{2}$$

(c) **3** (Maximum square error), which is given by the formula:

$$E_{\mathrm{MAX}} = \max_i \sqrt{\sum_{j=1}^{M}(y_{ij}-t_{ij})^2} \tag{3}$$

(d) **4** (A custom user-defined error function). With this value the user can specify as error function his own function. For this purpose he must alter the function user_function() in file *errorfunction.cc* of the distribution. This function is defined as
double user_function(**int** d, **int** n, **double** **origy, **double** **gdfy), where d is the dimension of the function to be estimated, n is the number of input patterns, *origy* is a two - dimensional table with dimensions $n \times d$ holding the values of the function to be estimated and *gdfy* is the values produced by the grammatical evolution procedure at the n specified points.

4 Experimental Results

The GDF2 program has been evaluated on a series of classification and regression datasets. These datasets are widely used for the evaluation of such methods and are presented below. They are also freely available for download from the Internet. In all cases, the train - test set ratio is 0.5 The datasets can be downloaded from the following URLs:

- http://www.ics.uci.edu/~mlearn/MLRepository.html
- http://www-ee.uta.edu/eeweb/IP/training_data_files.htm

The first URL contains the UCI repository of Machine Learning Databases and Domain Theories published under the Department of Information and

Computer Science, University of California. It contains 111 databases and domain theories covering a broad range of domains. The second URL contains 12 classification and regression related datasets from the Image Processing and Neural Networks Lab of the University of Texas.

Originally, the GDF tool was designed for solving only regression problems. However, with the recent multi-function parallel approximation addition, it is possible to use it for classification problems. This is logical since most classification methods create functions that optimally separate the different classes. In the next subsections a number of selected regression and classification datasets are presented. The experimental results that are produced using GDF are presented in the next section.

4.1 Regression datasets

The following regression datasets were used:

(1) **Expest**. This dataset contains data on parameter estimation on an exponential waveform function. The waveform is given by the equation $A \exp\left(\frac{-n}{T}\right) + n(n)$, where A and T have uniform densities between (1,4) and (10,50) respectively and n(n) is WGN with standard deviation of 0.1 The dataset contains 1000 data and the function to be estimated has a dimension of 2.
(2) **FMdemo**. The FMdemo Demodulation regression dataset contains data on signal frequency demodulation that contains a random waveform. The dataset contains 1000 examples of 7 attributes and the function to be estimated has dimension 1.
(3) **Building**. The Building dataset [5] was designed for a competition designed to test prediction algorithms. The dataset models energy consumption of a building, based on external environmental effects. It has 11 attributes: 8 inputs and 3 outputs. Four of the 8 inputs are used to encode the time at which the readings were taken: month, day, year and 4-digit hour and minute- all of which are discrete valued. The dataset covers the period of time between the 1st of September 1989 and 22nd February 1990, a total of 4182 patterns.
(4) **Oh7**. This data set is given in [6]. The training set contains VV and HH polarization at L 30, 40 deg, C 10, 30, 40, 50, 60 deg, and X 30, 40, 50 deg along with the corresponding unknowns rms surface height, surface correlation length, and volumetric soil moisture content in g / cubic cm. The function to be approximated has a dimension of 3.
(5) **Single2**. This training data file consists of 16 inputs and represents the training set for inversion of surface permittivity, the normalized surface rms roughness, and the surface correlation length found in back scattering

models from randomly rough dielectric surfaces. The first 16 inputs represent the simulated back scattering coefficient measured at 10, 30, 50 and 70 degrees at both vertical and horizontal polarization. The remaining 8 are various combinations of ratios of the original eight values. These ratios correspond to those used in several empirical retrieval algorithms.The function to be approximated has a dimension of 3 For more details, see [7].

(6) **Twod**. This training file is used in the task of inverting the surface scattering parameters from an inhomogeneous layer above a homogeneous half space, where both interfaces are randomly rough. The parameters to be inverted are the effective permittivity of the surface, the normalized rms height, the normalized surface correlation length, the optical depth, and single scattering albedo of an inhomogeneous irregular layer above a homogeneous half space from back scattering measurements. The training data file contains 1768 patterns. The inputs consist of eight theoretical values of back scattering coefficient parameters at V and H polarization and four incident angles. The outputs were the corresponding values of permittivity, upper surface height, lower surface height, normalized upper surface correlation length, normalized lower surface correlation length, optical depth and single scattering albedo which had a joint uniform pdf. The function to be approximated has a dimension of 7. For more details see [8].

4.2 Classification datasets

The classification datasets, that were used are the following:

(1) **Wine**. The wine recognition dataset contains data from wine chemical analysis. It contains 178 examples of 13 attributes. Each example may belong in one from three categories. The function to be approximated has a dimension of 1 and the acceptable values are 0 for the first class, 0.5 for the second class and 1.0 for the third class.

(2) **Class_Wine**. This dataset is the same as the previous, but the function to approximated has a dimension of 3. The output vector is a binary vector with value given by the equation:

$$t_{ij} = \begin{cases} 1, \text{ pattern i belongs to the class j} \\ 0, \qquad\qquad \text{otherwise} \end{cases}$$

(3) **Glass**. This dataset contains glass component analysis for glass pieces that belong to 6 classes. The dataset contains 214 examples with 10 attributes each. The function to be approximated has a dimension of 1 and the acceptable values are 0 for the first class, 0.166 for the second

Table 1
Regression Results with GDF2

DATASET	MSE_TRAIN	MSE_TEST	ABS_TRAIN	ABS_TEST
Expest	4.301	4.467	1.4929	1.5550
FMdemo	0.0516	0.0774	0.1660	0.2007
Building	0.0151	0.0150	0.1105	0.1102
Oh7	0.3902	0.4072	0.5290	0.5292
Single2	11.0876	10.4783	2.2217	2.1541
Twod	0.7665	0.8226	0.8057	0.8366

class, 0.333 for the third class, 0.667 for the fourth class, 0.833 for the fifth class and 1.0 for the sixth class.

(4) **Class_Glass**. This dataset is the same as the previous, but the function to be approximated has a dimension of six, with the same meaning as the dataset Class_Wine.

4.3 GDF2 Results

In this section, the experimental results for the above datasets are presented. They are divided into regression and classification results. In the classification case, thresholds have been used to create the classes. The GDF2 was tested in every dataset 30 times with different seed for the random numbers initialization each time using 500 chromosomes and 2000 generations. This was done in order to minimize the random variance effect from the data itself. The meaning of the columns in the following tables has as:

- MSE_TRAIN: Mean square train error.
- MSE_TEST: Mean square test error.
- ABS_TRAIN: Absolute square train error.
- ABS_TEST: Absolute square test error.
- CLASS_MSE: Mean square classification error in the test set.
- CLASS_ABS: Absolute square classification error in the test set.

The results from the application of GDF2 are shown in tables 1 and 2.

4.4 Neural Results

In table 3 we list the results from the application of a neural network with ten hidden units to the regression problems and in table 4 we list the results

Table 2
Classification results with GDF2

DATASET	CLASS_MSE	CLASS_ABS
Wine	13.62%	14.25%
Class_Wine	10.81%	12.79%
Glass	56.39%	51.56%
Class_Glass	45.83%	43.70%

Table 3
The results for the regression problems using a neural network

DATASET	TRAIN_MSE	TEST_MSE
Expest	27.266	25.301
FMdemo	0.005	0.0314
Building	0.009	0.009
Oh7	8.694	8.815
Single2	45.763	40.159
Two2	0.698	63.925

Table 4
The results for the classification problems using a neural network

DATASET	CLASS_MSE
Wine	55.70%
Class_Wine	51.44%
Glass	53.86%
Class_Glass	47.45%

from the application of the same neural network to the classification problems. The method BFGS [9] was used for the training of the neural network and the average of 30 independent runs is presented in the corresponding tables. The results show that the Neural network that is trained using BFGS, gives better results that the proposed method in half datasets for both regression and classification.

4.5 Rbf Results

In table 5 we list the results from the application of a Rbf network with ten hidden units to the regression problems and in table 6 we list the results from the application of the same neural network to the classification problems. The

Table 5
The results for the regression problems using an Rbf network

DATASET	TRAIN_MSE	TEST_MSE
Expest	38.63	727.30
FMdemo	0.039	0.127
Building	0.0331	0.165
Oh7	8.764	94.57
Single2	11.030	72.412
Two2	0.681	16.625

Table 6
The results for the classification problems using an Rbf network

DATASET	CLASS_MSE
Wine	25.44%
Class_Wine	20.04%
Glass	45.61%
Class_Glass	32.65%

results show that the Rbf network outperforms the proposed GDF2 algorithm in 50% of the regression datasets and in 50% of the classification datasets.

5 Test run output

Here we present three different outputs from experiments to the Expest dataset.

5.1 Experiment 1

In the first experiment we issue the following command under the *bin* directory:
./gdf -p expest.train -t expest.test -e 1
The output of the program in the last 5 generation is presented in figure 5.1.

5.2 Experiment 2

In the second experiment we issue the following command under the *bin* directory:

Fig. 2. The last five generations of the first experiment

```
fitness = -5.698335329
generation=495
f1(x)= (sqrt(x1))
f2(x)= x1+x1+23.04+(x2)*(6.0)+abs((9.98)^cos(x2))+sin(sqrt(x1+x2))/abs((log(sqrt(sin((2.7))))))
fitness = -5.698335329
generation=496
f1(x)= (sqrt(x1))
f2(x)= x1+x1+23.04+(x2)*(6.0)+abs((9.98)^cos(x2))+sin(sqrt(x1+x2))/abs((log(sqrt(sin((2.7))))))
fitness = -5.698335329
generation=497
f1(x)= (sqrt(x1))
f2(x)= x1+x1+23.04+(x2)*(6.0)+abs((9.98)^cos(x2))+sin(sqrt(x1+x2))/abs((log(sqrt(sin((2.7))))))
fitness = -5.698335329
generation=498
f1(x)= (sqrt(x1))
f2(x)= x1+x1+23.04+(x2)*(6.0)+abs((9.98)^cos(x2))+sin(sqrt(x1+x2))/abs((log(sqrt(sin((2.7))))))
fitness = -5.698335329
generation=499
f1(x)= (sqrt(x1))
f2(x)= x1+x1+23.04+(x2)*(6.0)+abs((9.98)^cos(x2))+sin(sqrt(x1+x2))/abs((log(sqrt(sin((2.7))))))
fitness = -5.698335329
generation=500
f1(x)= (sqrt(x1))
f2(x)= x1+x1+23.04+(x2)*(6.0)+abs((9.98)^cos(x2))+sin(sqrt(x1+x2))/abs((log(sqrt(sin((2.7))))))
fitness = -5.698335329
APPLICATION TO expest.test
TEST  ERROR=5.891387376
```

Fig. 3. The last five generations of the second experiment

```
fitness = -1.863288594
generation=495
f1(x)= log(x1)
f2(x)= x2+x1*log(exp((2.93)+sin(sqrt(x1))))+abs((27.53/28.1+exp(x2)-(x1^sqrt(1.00)/x1))*40.6)
fitness = -1.863288594
generation=496
f1(x)= log(x1)
f2(x)= x2+x1*log(exp((2.93)+sin(sqrt(x1))))+abs((27.53/28.1+exp(x2)-(x1^sqrt(1.00)/x1))*40.6)
fitness = -1.863288594
generation=497
f1(x)= log(x1)
f2(x)= x2+x1*log(exp((2.93)+sin(sqrt(x1))))+abs((27.53/28.1+exp(x2)-(x1^sqrt(1.00)/x1))*40.6)
fitness = -1.863288594
generation=498
f1(x)= log(x1)
f2(x)= x2+x1*log(exp((2.93)+sin(sqrt(x1))))+abs((27.53/28.1+exp(x2)-(x1^sqrt(1.00)/x1))*40.6)
fitness = -1.863288594
generation=499
f1(x)= log(x1)
f2(x)= x2+x1*log(exp((2.93)+sin(sqrt(x1))))+abs((27.53/28.1+exp(x2)-(x1^sqrt(1.00)/x1))*40.6)
fitness = -1.863288594
generation=500
f1(x)= log(x1)
f2(x)= x2+x1*log(exp((2.93)+sin(sqrt(x1))))+abs((27.53/28.1+exp(x2)-(x1^sqrt(1.00)/x1))*40.6)
fitness = -1.863288594
APPLICATION TO expest.test
TEST  ERROR=1.769045368
```

./gdf -p expest.train -t expest.test -e 2

In this experiment we use the ABS_FUNCTION of equation (1) as the error function. The output of the program in the last 5 generation is presented in figure 5.2.

5.3 Experiment 3

In this experiment we extend our grammar with 3 UDFs. The first UDF is the sigmoidal function, defined by the equation:

$$\mathrm{UDF}_1(x) = \frac{1}{1+\exp(-x)}$$

The second UDF is the step function, defined by the equation:

$$\mathrm{UDF}_2(x) = \begin{cases} 1, & x > 0 \\ 0, \text{ otherwise} \end{cases}$$

The third UDF is a gaussian function, defined by the equation:

$$\mathrm{UDF}_3(x) = \exp\left(\frac{-x^2}{2}\right)$$

These functions are defined in the file *udf.cc* under the subdirectory *src* of the distribution. The user can alter the file, according to the needs of his experiment. We issue the command:
./gdf -p expest.train -t expest.test -e 2 -g udf_grammar.txt
The parameter -g is needed, in order to use the extended grammar. The output of the program in the last 5 generation is presented in figure 5.3.

Acknowledgements

All the experiments of this report were executed at the Research Center for Scientific Simulations of the University of Ioannina, which is composed of 200 computing nodes with dual cpus (AMD OPTERON 2.2GHZ 64bit) running RedHat Enterprise Linux.

References

[1] M. O'Neill and C. Ryan, Grammatical Evolution, IEEE Trans. Evolutionary Computation 5 (2001), pp. 349-358.

Fig. 4. The last five generations of the third experiment

```
fitness = -1.657529401
generation=495
f1(x)= log(x1)+udf1(sin(x2))
f2(x)= x1+exp((abs(((7.48)+x2)))^sin((log(2.08)-x1+x1*udf1((x1)))))

fitness = -1.657529401
generation=496
f1(x)= log(x1)+udf1(sin(x2))
f2(x)= x1+exp((abs(((7.48)+x2)))^sin((log(2.08)-x1+x1*udf1((x1)))))

fitness = -1.657529401
generation=497
f1(x)= log(x1)+udf1(sin(x2))
f2(x)= x1+exp((abs(((7.48)+x2)))^sin((log(2.08)-x1+x1*udf1((x1)))))

fitness = -1.657529401
generation=498
f1(x)= log(x1)+udf1(sin(x2))
f2(x)= x1+exp((abs(((7.48)+x2)))^sin((log(2.08)-x1+x1*udf1((x1)))))

fitness = -1.657529401
generation=499
f1(x)= log(x1)+udf1(sin(x2))
f2(x)= x1+exp((abs(((7.48)+x2)))^sin((log(2.08)-x1+x1*udf1((x1)))))

fitness = -1.657529401
generation=500
f1(x)= log(x1)+udf1(sin(x2))
f2(x)= x1+exp((abs(((7.48)+x2)))^sin((log(2.08)-x1+x1*udf1((x1)))))

fitness = -1.657529401
APPLICATION TO expest.test
TEST  ERROR=1.558419354
```


[2] H. Ishibuchi and T. Murata, "Multi-objective genetic local search algorithm", in Proceedings of the 1966 IEEE International Conference on Evolutionary Computation, IEEE Press, NJ, 1996, pp. 119-124.

[3] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, pp. 182-197, 2002.

[4] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the Strength Pareto approach", IEEE Transactions on Evolutionary Computation, vol.3, no. 4, pp. 257-271, 1999.

[5] "The Great Energy Predictor Shootout" - The First Building Data Analysis And Prediction Competition, Co-chaired by Jan F. Kreider and Jeff S. Haberl, Denver, Colorado.

[6] Oh, Y., K. Sarabandi, and F.T. Ulaby, "An Empirical Model and an Inversion Technique for Radar Scattering from Bare Soil Surfaces," IEEE Trans. on Geoscience and Remote Sensing, pp. 370-381, 1992.

[7] A. K. Fung, Z. Li, and K. S. Chen, "Back scattering from a Randomly Rough Dielectric Surface," IEEE Trans. Geo. and Remote Sensing, Vol. 30, No. 2, March 1992. A. K. Fung, Microwave Scattering and Emission Models and Their Applications, Arctec House, 1994

[8] M. S. Dawson, A. K. Fung and M. T. Manry, "Surface parameter retrieval using fast learning neural networks," Remote Sensing Reviews, 1993, Vol. 7(1), pp. 1-18. M. S. Dawson, J. Olvera, A. K. Fung and M. T. Manry, "Inversion of surface parameters using fast learning neural networks," Proc. of IGARSS'92, Houston, Texas, May 1992, Vol II, pp 910 912.

[9] M. J. D. Powell, " A tolerant algorithm for Linearly Constrained Optimization Calculations", Mathematical Programming 45 (1989), 547.