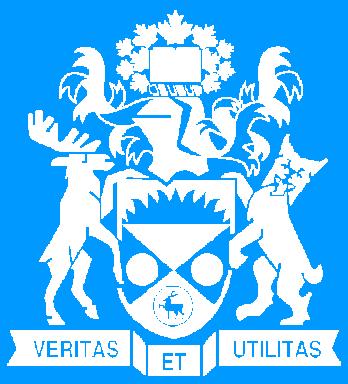
THE UNIVERSITY OF WESTERN ONTARIO
DEPARTMENT OF CIVIL AND
ENVIRONMENTAL ENGINEERING

Water Resources Research Report

Optimization Using Differential Evolution

By:
Vasan Arunachalam



Report No: 060
Date: July 2008

ISSN: (print) 1913-3200; (online) 1913-3219;
ISBN: (print) 978-0-7714-2689-6; (online) 978-0-7714-2690-2;

OPTIMIZATION USING DIFFERENTIAL EVOLUTION

By

VASAN ARUNACHALAM

Facility for Intelligent Decision Support
Department of Civil and Environmental Engineering
The University Of Western Ontario, London, Ontario, Canada

JULY 2008

ABSTRACT

The book explains in detail the working of Differential Evolution optimization algorithm. It also provides documentation for the use of Differential Evolution computer program to solve user-defined optimization problems. The computer program is written in C language for Windows environment. The book also demonstrates how to modify the program using an example optimization problem.

This source code is distributed for academic purposes only. It has no warranty implied or given, and the author assumes no liability for damage resulting from its use or misuse. The author can be contacted for any comments by email at vasan.arunachalam@gmail.com.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Optimization is a procedure through which the best possible values of decision variables are obtained under the given set of constraints and in accordance to a selected optimization objective function. The most common optimization procedure applies to a design that will minimize the total cost or maximize the possible reliability or any other specific objective. Fields of science and engineering, business decision-making and industry are all rich in problems that require the implementation of optimization approach. Since, most real world optimization problems seem to be both fundamentally and practically hard, research into better algorithms remains valuable and continues, so that, one can guarantee to find the best solution using an efficient optimization algorithm.

Nowadays, there exist a lot of optimization algorithms that work using gradient-based and heuristic-based search techniques in deterministic and stochastic contexts. In order to widen the applicability of the optimization approach to various problem domains, natural and physical principles are mimicked to develop robust optimization algorithms. Evolutionary algorithms, simulated annealing, ant colony optimization, memetic algorithms, particle swarm optimization are few examples of such algorithms.

Over the last decade, evolutionary algorithms have been extensively used in various problem domains and succeeded in effectively finding the near optimal solutions. The present book provides a detailed description of one such evolutionary algorithm, named, *Differential Evolution (DE).* Since its inception in 1995, DE has earned a reputation of a very effective global optimizer (Storn and Price, 1995). Chapter 1 begins with an introduction to the optimization and Chapter 2 introduces the detailed working of DE

algorithm with an illustrative example. Chapter 3 provides the instructions for using the DE computer program developed for Windows environment that accompanies this book on CD-ROM. Chapter 4 concludes with few practical suggestions for the users of the DE computer program. An Appendix to the text contains description of the source code of DE software developed in C programming language.

CHAPTER 2

DIFFERENTIAL EVOLUTION

WORKING OF THE DE ALGORITHM

Differential Evolution (DE) algorithm is a branch of evolutionary programming developed by Rainer Storn and Kenneth Price (Price and Storn, 1997) for optimization problems over continuous domains. In DE, each variable's value is represented by a real number. The advantages of DE are its simple structure, ease of use, speed and robustness. DE is one of the best genetic type algorithms for solving problems with the real valued variables. Differential Evolution is a design tool of great utility that is immediately accessible for practical applications. DE has been used in several science and engineering applications to discover effective solutions to nearly intractable problems without appealing to expert knowledge or complex design algorithms. If a system is amenable to being rationally evaluated, DE can provide the means for extracting the best possible performance from it. Differential Evolution uses *mutation* as a search mechanism and *selection* to direct the search toward the prospective regions in the feasible region. Genetic Algorithms generate a sequence of populations by using *selection mechanisms*. Genetic Algorithms use *crossover* and *mutation* as search mechanisms. The principal difference between Genetic Algorithms and Differential Evolution is that Genetic Algorithms rely on *crossover*, a mechanism of probabilistic and useful exchange of information among solutions to locate better solutions, while evolutionary strategies use *mutation* as the primary search mechanism.

DE is a population based search technique which utilizes *NP* variables as population of *D* dimensional parameter vectors for each generation. The initial population is chosen

randomly if no information is available about the problem. In the case of the available preliminary solution, the initial population is often generated by adding normally distributed random deviations to the preliminary solution. The basic idea behind DE is a new scheme for generating trial parameter vectors. DE generates new parameter vectors by adding the weighted difference vector between two population members to a third member. If the resulting vector yields a lower objective function value than a predetermined population member, the newly generated vector replaces the vector with which it was compared. In addition, the best parameter vector is evaluated for every generation in order to keep track of the progress that is made during the optimization process. Extracting the distance and the direction information from the population to generate random deviations result in an adaptive scheme with excellent convergence properties (Price et al., 2005).

DE maintains two arrays, each of which holds a population size *NP* and *D* dimensional, real-valued vectors. The primary array holds the current vector population, while the secondary array accumulates vectors that are selected for the next generation. In each generation, *NP* competitions are held to determine the composition of the next generation.

Every pair of vectors (X_a, X_b) defines a vector differential: $(X_a - X_b)$. When X_a and X_b are chosen randomly, their weighted differential is used to perturb another randomly chosen vector X_c. This process can be mathematically expressed as:

$$X_c' = X_c + F(X_a - X_b) \qquad (1)$$

The weighting, or scaling, factor *F* is a user supplied constant in the optimal range between 0.5 and 1.0 (DE, 2008). In every generation, each primary array vector X_i is targeted for crossover with a vector like X_c' to produce a trial vector X_t. Thus, the trial vector is the child of two parents, a noisy random vector and the target vector against which it must compete. *Uniform crossover* (that can take child vector parameters from

one parent more often than it does from others) is used with a crossover constant (*CR*), in the optimal range of 0.5 to 1.0 (DE, 2008) which actually represents the probability that the child vector inherits the parameter values from the noisy random vector. When *CR* = 1, for example, every trial vector parameter is certain to come from X_c'. On the other hand, if *CR* = 0, all but one trial vector parameter comes from the target vector. To ensure that X_t differs from X_i by at least one parameter, the final trial vector parameter always comes from the noisy random vector even when *CR* = 0. Then the objective function corresponding to the trial vector is compared with that of the target vector, and the vector that has the lower objective function value (for minimization) of the two would survive for the next generation. This process is continued until the termination criterion of a preset maximum number of generations (*MAXGEN*) is met, and difference in objective function values between two consecutive generations reaches a small value. Figure 1 shows the working of the DE algorithm.

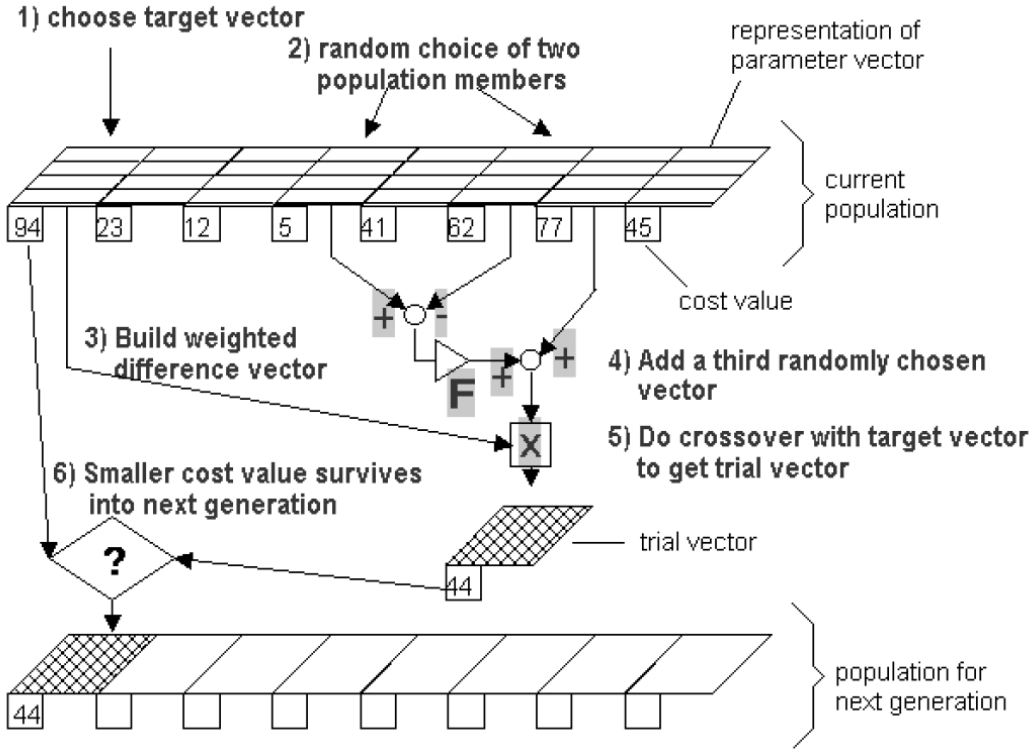


Fig. 1. Working of Differential Evolution Algorithm

Price & Storn (1997) gave the working principle of DE with single strategy. Later on, they suggested ten different strategies for DE. Different strategies can be adopted in the DE algorithm depending upon the type of problem to which DE is applied. The strategies can vary based on the vector to be *perturbed*, number of difference vectors considered for perturbation, and finally the type of *crossover* used. The following are the ten different working strategies:

1. DE/best/1/exp
2. DE/rand/1/exp
3. DE/rand-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/rand-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

The general convention used above is *DE/x/y/z*. *DE* stands for Differential Evolution, *x* represents a string denoting the vector to be perturbed, *y* is the number of difference vectors considered for perturbation of *x*, and *z* stands for the type of *crossover* being used (exp: exponential; bin: binomial). Hence the *perturbation* can be either in the best vector of the previous generation or in any randomly chosen vector. Similarly for *perturbation* either single or two vector differences can be used. For *perturbation* with a single vector difference, out of the three distinct randomly chosen vectors, the weighted vector differential of any two vectors is added to the third one. Similarly for *perturbation* with two vector differences, five distinct vectors, other than the target vector are chosen randomly from the current population. Out of these, the weighted vector difference of each pair of any four vectors is added to the fifth one for perturbation. In *exponential crossover,* the crossover is performed on the D variables in one loop until it is within the *CR* bound. The first time a randomly picked number between 0 and 1 goes beyond the *CR* value, no *crossover* is performed and the remaining D variables are left intact. In

binomial crossover, the crossover is performed on each of the D variables whenever a randomly picked number between 0 and 1 is within the *CR* value. So for high values of *CR*, the *exponential and binomial crossover* methods yield similar results.

A strategy that works out to be the best for a given problem may not work well when applied to a different problem. Also, the strategy and the key parameters to be adopted for a problem are to be determined by trial and error. However, strategy-7 (DE/rand/1/bin) appears to be the most successful and the most widely used strategy. In all, three factors control evolution under DE, the population size *NP*, the weight applied to the random differential *F* and the crossover constant *CR*. More details regarding DE are available in Price and Storn (1997), Onwubolu and Babu (2004) and Price and Storn (2005).

AN ILLUSTRATIVE EXAMPLE

A simple numerical example is presented to illustrate the DE algorithm. Let us consider the following objective function:

$$Minimize \quad f(x) = x_1 + x_2 + x_3 \tag{2}$$

The initial population is generated (chosen randomly between the bounds of decision variables, in this case within 0 and 1, for the three decision variables. The population along with its respective objective function values is shown in Table 1. The first member of the population "Individual 1" is set as the target vector.

In order to generate the noisy random vector, three individuals (Individual 2, Individual 4 and Individual 6) from the population size are selected randomly (ignoring "Individual 1", since it is set as the target vector). The weighted difference between "Individual 2" and "Individual 4" is added to the third randomly chosen vector "Individual 6" to generate the noisy random vector. The weighting factor *F* is chosen as 0.80 and the weighted difference vector is obtained in Table 2 and the noisy random vector in Table 3.

Table 1. An illustrative example

	Population Size $NP = 6$ (user defined), $D = 3$					
	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6
x_1	0.68	0.92	0.22	0.12	0.40	0.94
x_2	0.89	0.92	0.14	0.09	0.81	0.63
x_3	0.04	0.33	0.40	0.05	0.83	0.13
$f(x)$	1.61	2.17	0.76	0.26	2.04	1.70

Table 2. Calculation of the weighted difference vector for the illustrative example

	Individual 2		Individual 4	Difference Vector		Weighted Difference Vector
x_1	0.92		0.12	= 0.80		= 0.64
x_2	0.92	-	0.09	= 0.83	$\times F$ ($F = 0.80$)	= 0.66
x_3	0.33		0.05	= 0.28		= 0.22

Table 3. Calculation of the noisy random vector for the illustrative example

	Weighted Difference Vector		Individual 6	Noisy Random Vector
x_1	0.64		0.94	= 1.58
x_2	0.66	+	0.63	= 1.29
x_3	0.22		0.13	= 0.35

The noisy random vector does a *crossover* with the target vector to generate the trial vector as shown in Table 4. This is carried out by (1) generating random numbers equal to the dimension of the problem (2) for each of the dimensions: if random number > *CR*; copy the value from the target vector, else copy the value from the noisy random vector into the trial vector. In this example, the crossover constant *CR* is chosen as 0.50.

Table 4. Generation of the trial vector for the illustrative example

	Target Vector		Noisy Random Vector	Trial Vector
x_1	0.68	Crossover (CR = 0.50)	1.58	= 1.58
x_2	0.89		1.29	= 0.89
x_3	0.04		0.35	= 0.04
$f(x)$	1.61		3.22	2.51

The objective function of the trial vector is compared with that of the target vector and the vector with the lowest value of the two becomes "Individual 1" for the next generation. To evolve "Individual 2" for the next generation, the second member of the population is set as target vector and the above process is repeated. This process is repeated NP times till the new population set array is filled which completes one generation. Once the termination criterion is met, the algorithm ends.

Table 5. New population for next generation for the illustrative example

	New Population for Next Generation					
	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6
x_1	0.68					
x_2	0.89					
x_3	0.04					
$f(x)$	1.61					

CHAPTER 3

HOW TO RUN THE DE PROGRAM?

Single objective Differential Evolution optimization algorithm has been programmed in C language using Microsoft Visual C++ 6.0 for Windows. This Chapter explains how to use the DE computer program for solving any optimization problem. The source code of the DE program is available in Appendix I. The CD enclosed along with this book contains a folder named "Program". The folder contains the C program file "DE.c" which requires two input files, "Limits.txt" and "ParameterLimits.txt". The program generates two output files, "ReportDE.html" and "Convergence.txt".

In order to learn how to use the DE program, consider the following optimization problem:

$$\text{Minimize} \quad f(x) = \left(x_1^{\,2} + x_2 - 11\right)^2 + \left(x_1 + x_2^{\,2} - 7\right)^2$$

subject to

$$\left(x_1 - 5\right)^2 + x_2^{\,2} - 26 \le 0$$
$$4x_1 - x_2 - 20 \le 0$$
$$x_1, x_2 \ge 0$$

The algorithm is designed to accept optimization problem with an objective of minimization type. If the objective of the problem is of maximization type, it should be modified accordingly to use in the program. Each decision variable is represented from x[0], x[1], x[2] and so on. Accordingly, the objective function for the example is written as follows:

```
(x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11)+(x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7)
```

To enter the constraints, the inequality constraints should be modified to the less or equal format, $g(x) \leq 0$. If the problem is an unconstrained optimization problem, the user need not enter anything in the space specified for the constraints coding. The constraints of the example are entered as follows:

```
/* Constraint 1*/
temp = (x[0]-5)*(x[0]-5) + x[1]*x[1] - 26;
val[1] = max(temp,0);

/* Constraint 2 */
temp = 4*x[0] + x[1] -20;
val[2] = max(temp,0);
```

If the first constraint of the example is an equality constraint, the constraint should be entered as follows:

```
/* Constraint 1 (Equality Condition) */
val[1] = abs((x[0]-5)*(x[0]-5) + x[1]*x[1] – 26);
```

where val[1] and val[2] represent the amount of violation for each constraint. These values are multiplied by a penalty co-efficient penal[1] and penal[2] (entered by the user of the program), which is then added to the objective function to continue the process of optimization. This process is often termed as penalty function approach. A simple additive penalty function approach is used in order to convert the constrained problem into unconstrained problem. Due to this conversion, the solution falling outside the feasible region is penalized and forced to fall into the feasible solution space after a few generations. However, the penalty function method has certain weaknesses that are fatal when the penalty parameters are large. Penalty functions tend to be ill conditioned near the boundary of the feasible domain and that may result in a local optimal solution or an

infeasible solution. Careful selection of the penalty parameters is required for the proper convergence to a feasible optimal solution.

The main C program has many user-defined functions. The user needs to enter objective function and constraints in the user-defined function evaluate( ). The program code is commented, so the user needs to follow the comments to enter the number of constraints, ncons (in this example, ncons = 2), objective function, constraints (if any) and the penalty coefficient for each constraint (if any), at the appropriate places. The "evaluate( )" function for the example problem is displayed below:

```
double evaluate(double *x)
{
        int ncons, i;
        double *val, *penal, temp;
        long double value=0;
        feval++;
        /*   DO NOT CHANGE ANYTHING ABOVE   */
        /* Enter the Number of Constraints to the variable ncons*/
        ncons = 2;
        /*-------------------------------------------------------*/
        val      = (double *)malloc(sizeof(double)*ncons);
        penal    = (double *)malloc(sizeof(double)*ncons);
        penal[0]=1;

        /*----------------------CODE YOUR OBJECTIVE FUNCTIONS HERE------------*/
        /*All functions must be of minimization type*/
        /*============Start Coding Your Objective Function=============*/
        val[0] = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) +
                 (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7);
        /*========END YOUR CODING UPTO THIS POINT============*/
```

```
        /*----------------------CODE YOUR CONSTRAINTS HERE------------*/
         /*All functions must be of minimization type, negate maximization functions */
        /*Constraints must be of the following type: g(x) <= 0. Enter the constraint
        equation to temp variable and don't change the expression for val[1].
        Similarly, do the same for other constraints. */
        /*============Start Coding Your Constraints=============*/
        /* Constraint 1*/
        temp = (x[0]-5)*(x[0]-5) + x[1]*x[1] - 26;
        val[1] = max(temp,0);

        /* Constraint 2*/
        temp = 4*x[0] + x[1] -20;
        val[2] = max(temp,0);

        /*Enter the Penalty Coefficient for Each constraint*/
        penal[1] = 1000;
        penal[2] = 1000;
        /*========END YOUR CODING UPTO THIS POINT============*/
        for (i=0;i<=ncons;i++) value = (value + penal[i]*val[i]);
        return value;
}
```

Similarly, the objective function without constraints needs to be entered in the function "evaluateWOpenalty( )". The "evaluateWOpenalty( )" function for the example problem is displayed below:

```
double evaluateWOpenalty(double *x)
{
        double value;
        /*   DO NOT CHANGE ANYTHING ABOVE   */
```

```
        /*--------------------CODE YOUR OBJECTIVE FUNCTIONS HERE------------*/
        /*All functions must be of minimization type*/
        /*============Start Coding Your Objective Function=============*/

        value = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) +
                (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-7);
        /*=========END CODING UPTO THIS POINT==================*/
        return value;
}
```

After inserting the objective function and constraints, the user needs to prepare the two input files ("Limits.txt" and "ParameterLimits.txt"). In the "Limits.txt" input file, the lower and upper bound for each decision variable separated by a tab, is entered. The number of decision variables for the example is two. The lower and upper bounds for each variable is assumed as 0 and 10 respectively. The "Limits.txt" input file for the example is as follows:

0	10	→	Lower and Upper bound of first decision variable
0	10	→	Lower and Upper bound of second decision variable

The "ParameterLimits.txt" input file requires the number of decision variables (in the first row), maximum number of generations (in the second row), minimum and maximum number of population (*NP)*, crossover constant (*CR*), weighting factor (*F*) along with their step length for sensitivity analysis in the third, fourth and fifth rows respectively. The input file for the example problem is as follows:

2			→	Number of decision variables
30			→	Maximum number of generations
20	20	10	→	Minimum, maximum and step length for *NP*
0.8	0.9	0.1	→	Minimum, maximum and step length for *CR*
0.5	0.6	0.1	→	Minimum, maximum and step length for *F*

This completes the preparation of inputs files. To generate the optimal set of solutions for the optimization problem, DE program is compiled and executed. The program can be compiled and execution using any standard C compiler for Windows environment. After the execution starts, a DOS window pops up (as shown in Figure 2), displaying the progress of the optimization process.

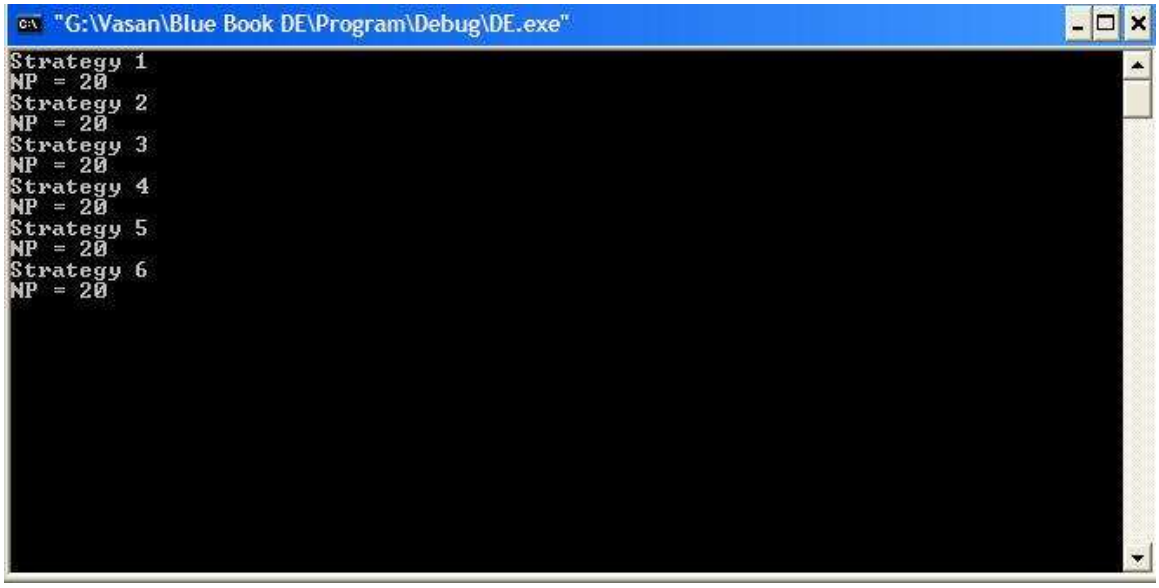


Fig. 2. DOS window displaying the progress of optimization

When the program is run for different combinations of *NP, CR* and *F*, the optimal set of parameters is determined based on two factors i.e., minimum objective function value and lower CPU time requirement. In any given situation, if minimum objective function values are the same for any given combination(s), the next criteria that is chosen for selecting optimal combination is lower CPU time requirement. In this program, these two factors are considered for choosing optimal set of parameters. Two output files ("ReportDE.html" and "Convergence.txt") are generated after the successful completion of the program's execution.

"ReportDE.html" output file prints the detailed record of the objective function value for all combinations of *NP, CR* and *F* for all ten strategies. It also records the best ever combination for each strategy along with its objective function value, constraint

violation, number of function evaluations and computational time. The optimal value of objective function and decision variables for the optimization problem is also recorded. The output file for the example is as follows:

Optimization by Differential Evolution

Strategy 1 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	1.776225E-005 (31 ms)	8.532469E-005 (62 ms)
CR = 0.90	1.954033E-006 (63 ms)	1.214437E-004 (78 ms)

Strategy 2 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	8.589940E-010 (78 ms)	7.677683E-009 (78 ms)
CR = 0.90	3.381982E-008 (94 ms)	7.682233E-007 (94 ms)

Strategy 3 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	3.777920E-005 (78 ms)	3.098915E-005 (94 ms)
CR = 0.90	4.402272E-007 (93 ms)	1.464849E-004 (94 ms)

Strategy 4 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	2.198994E-004 (78 ms)	5.341329E-007 (110 ms)
CR = 0.90	7.713227E-006 (109 ms)	1.041404E-004 (78 ms)

Strategy 5 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	5.074285E-007 (94 ms)	4.970142E-007 (94 ms)
CR = 0.90	5.250883E-005 (93 ms)	1.778123E-007 (94 ms)

Strategy 6 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	2.001744E-007 (110 ms)	6.724820E-005 (93 ms)
CR = 0.90	8.234626E-008 (125 ms)	3.239539E-005 (94 ms)

Strategy 7 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	7.588175E-009 (94 ms)	9.897662E-008 (94 ms)
CR = 0.90	7.055386E-009 (93 ms)	2.659110E-007 (94 ms)

Strategy 8 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	1.362700E-006 (109 ms)	1.225594E-005 (110 ms)
CR = 0.90	4.150216E-008 (109 ms)	7.398449E-006 (94 ms)

Strategy 9 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	5.604798E-006 (109 ms)	4.277174E-004 (94 ms)
CR = 0.90	3.914301E-005 (109 ms)	3.000180E-004 (94 ms)

Strategy 10 :
NP = 20

NP = 20	F = 0.50	F = 0.60
CR = 0.80	1.259527E-009 (110 ms)	8.828525E-009 (93 ms)
CR = 0.90	1.220839E-006 (94 ms)	3.677681E-007 (94 ms)

Results :

Strategy No.	Strategy	NP	CR	F	Optimal Value	Constraint Violation	NFE	Time Taken(ms)
1	DE/rand/1/bin	20	0.90	0.50	1.954033E-006	0.0000E+000	703	63
2	DE/best/1/bin	20	0.80	0.50	8.589940E-010	0.0000E+000	681	78
3	DE/best/2/bin	20	0.90	0.50	4.402272E-007	0.0000E+000	681	93
4	DE/rand/2/bin	20	0.80	0.60	5.341329E-007	0.0000E+000	769	110
5	DE/rand-to-best/1/bin	20	0.90	0.60	1.778123E-007	0.0000E+000	681	94
6	DE/rand/1/exp	20	0.90	0.50	8.234626E-008	0.0000E+000	879	125
7	DE/best/1/exp	20	0.90	0.50	7.055386E-009	0.0000E+000	681	93
8	DE/best/2/exp	20	0.90	0.50	4.150216E-008	0.0000E+000	725	109
9	DE/rand/2/exp	20	0.80	0.50	1.954033E-006	0.0000E+000	747	109
10	DE/rand-to-best/1/exp	20	0.80	0.50	1.259527E-009	0.0000E+000	769	110

Best Strategy is Strategy **DE/rand/1/bin**

Minimum constraint violation (CV) : **0.0000E+000**
Minimum objective value with min CV: **1.954033E-006**
Minimum time taken : **63**

Parameter	Value
X_1	3.000126
X_2	2.000219

End of report

"Convergence.txt" output file prints the record of the best objective function value for each generation, strategy wise.

The maximum number of generations for the program has been limited to 5000. If the user requires to run the program for more than 5000 generations, a minor modification in the program is necessary. The first line in the main( ) program assigns the variable nx=5000. The value of the variable nx needs to be modified accordingly. Similarly, the value of EPSILON (i.e., the difference in objective function values between two consecutive generations) can also be changed by the user in the program. It can be found in the 17th line of the program as "#define EPSILON 0.000001". The default value of

EPSILON is set as 0.000001 which is good enough for all kinds of optimization problems.

Thus, one can modify the program to determine the optimal solution for any optimization problem. This source code is distributed for academic purposes only. It has no warranty implied or given, and the author assumes no liability for damage resulting from its use or misuse. Please forward comments or question to the author at vasan.arunachalam@gmail.com.

CHAPTER 4

CONCLUSION

The book explains optimization using the Differential Evolution (DE) algorithm in detail. The DE computer program is available on the accompanying CD. The source code is written in C language in Microsoft Visual C++ 6.0 environment and a basic knowledge of C programming language is sufficient to modify the program. The explanations on how to modify the DE code to use it for various optimization problems are provided. The use of the DE computer program is illustrated using an example problem.

REFERENCES

DE. "*Differential Evolution Homepage*." <http://www.icsi.berkeley.edu/~storn/code.html> (Last accessed on July 15, 2008).

Onwubolu, G. C., and Babu, B. V. (2004). *New Optimization Techniques in Engineering*, Springer-Verlag, Germany.

Price, V. Kenneth., and Storn, M. Rainer. (1997). "Differential evolution - A simple evolution strategy for fast optimization." *Dr. Dobb's Journal*, 22, 18-24 and 78.

Price, V. Kenneth., Storn, M. Rainer., and Lampinen, A. Jouni. (2005). *Differential evolution: A practical approach to global optimization*. Springer-Verlag Berlin, Heidelberg.

Storn, R., Price, V. Kenneth. (1995). *Differential Evolution – A simple and eifficient adaptive scheme for global optimization over continous spaces*. Technical Report TR-95-012, ICSI.

APPENDIX A
SOURCE CODE OF DE ALGORITHM IN C

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/timeb.h>
#include <time.h>
#include <math.h>
double sumarray(double *x,int start,int end);
double evaluateWOpenalty(double *x);
double evaluate(double *x);
double getDouble(const char *prompt);
double rnd_uni();
int getInt(const char *prompt);
void copyPoint(double *dest,double *source,int D);
double optimise(int gen_max, int D, int NP,double F,double CR,int strategy,double *Point);
int feval;
long timeTaken;
double **x1, **x2, resultpen;
#define EPSILON 0.000001
#define startBold fprintf(fp,"\n");fprintf(fp,"<font face=\"Arial\" size=+1>")
#define endBold fprintf(fp,"\n");fprintf(fp,"</font>")
#define doubleLine fprintf(fp,"<BR><BR>\n")
#define singleLine fprintf(fp,"<BR>\n")
#define Heading1 fprintf(fp,"<font face=\"Arial\" size=2>\n")
#define endHeading1 fprintf(fp,"</font><BR>\n")
#define Heading2 fprintf(fp,"<font face=\"Arial\" size=2>\n")
#define endHeading2 fprintf(fp,"</font><BR>\n")

/* HTML Report Generation functions */
void startHTMLBODY(FILE *fp)
{
        fprintf(fp,"<HTML><HEAD>");
        fprintf(fp,"<style type=\"text/css\"><!--\
.borderTL {\
        border-top: 1px solid #cccccc;\
        border-right: 1px none #999999;\
        border-bottom: 1px none #999999;\
        border-left: 1px solid #cccccc;\
}\
.borderBR {\
        border-top: 1px none #e9e9e9;\
        border-right: 1px solid #cccccc;\
```

```
        border-bottom: 1px solid #cccccc;\
        border-left: 1px none #e9e9e9;\
}\
.data {\
        font-family: Arial, Helvetica, sans-serif;\
        font-size: 11px;\
        vertical-align: middle;\
        border-top-width: 1px;\
        border-right-width: 1px;\
        border-bottom-width: 1px;\
        border-left-width: 1px;\
        border-top-style: none;\
        border-right-style: solid;\
        border-bottom-style: solid;\
        border-left-style: none;\
        border-top-color: #999999;\
        border-right-color: #cccccc;\
        border-bottom-color: #cccccc;\
        border-left-color: #999999;\
        width: 200px;\
}\
.data1 {\
        font-family: Arial, Helvetica, sans-serif;\
        font-size: 11px;\
        text-align: center;\
        vertical-align: middle;\
        border-top-width: 1px;\
        border-right-width: 1px;\
        border-bottom-width: 1px;\
        border-left-width: 1px;\
        border-top-style: none;\
        border-right-style: solid;\
        border-bottom-style: solid;\
        border-left-style: none;\
        border-top-color: #999999;\
        border-right-color: #cccccc;\
        border-bottom-color: #cccccc;\
        border-left-color: #999999;\
        width: 150px;\
}\
-->\
</style>\n");
        fprintf(fp,"<TITLE>Report generated</TITLE>\n</HEAD>\n<BODY>\n");
        fprintf(fp,"<FONT face=\"Arial\" size=2>\n");
        fprintf(fp,"<FONT face=\"Arial\" size=+3 color=\"#9999dd\">\n");
        fprintf(fp,"Optimization by Differential Evolution");
        fprintf(fp,"</FONT><BR><HR><BR>");
        return;
}
```

```
void closeHTML(FILE *fp)
{
	fprintf(fp,"\n</FONT>\n</BODY>\n</HTML>\n");
	return;
}

int main()
{
	int GENMAX, PARAMS, NP, NPmin, NPmax, sNP, i, j, nx=5000, ny, mx=10, oNP[10],
	oNFE[10], strategy, best=0;
	double CR, F, result, CRmin, CRmax, sCR, Fmin, Fmax, sF, *oCR, *oF, *oResult,
	*wResult, **oPoint, *Point;
	long *oTime;
	char *strategies[14] = {
		"DE/rand/1/bin",
		"DE/best/1/bin",
		"DE/best/2/bin",
		"DE/rand/2/bin",
		"DE/rand-to-best/1/bin",
		"DE/rand/1/exp",
		"DE/best/1/exp",
		"DE/best/2/exp",
		"DE/rand/2/exp",
		"DE/rand-to-best/1/exp"
	};
	FILE *fp;

	fp	= fopen("ParameterLimits.txt","r");
	fscanf(fp,"%d",&PARAMS);
	fscanf(fp,"%d",&GENMAX);
	fscanf(fp,"%d %d %d",&NPmin,&NPmax,&sNP);
	fscanf(fp,"%lf %lf %lf",&CRmin,&CRmax,&sCR);
	fscanf(fp,"%lf %lf %lf",&Fmin,&Fmax,&sF);
	fclose(fp);

	ny=PARAMS;

	Point	= (double *)malloc(sizeof(double)*ny);
	oCR		= (double *)malloc(sizeof(double)*mx);
	oF		= (double *)malloc(sizeof(double)*mx);
	oResult = (double *)malloc(sizeof(double)*mx);
	wResult	= (double *)malloc(sizeof(double)*mx);
	oTime	= (long *)malloc(sizeof(long)*mx);

	x1  = (double **) malloc(nx * sizeof(double*));
	x1[0]  = (double *) malloc((nx * ny) * sizeof(double));
	for(i = 1; i < nx; i++ ) x1[i]  = x1[i-1] + ny;

	x2  = (double **) malloc(nx * sizeof(double*));
	x2[0]  = (double *) malloc((nx * ny) * sizeof(double));
	for(i = 1; i < nx; i++ ) x2[i]  = x2[i-1] + ny;
```

```
oPoint  = (double **) malloc(mx * sizeof(double*));
oPoint[0]  = (double *) malloc((mx * ny) * sizeof(double));
for(i = 1; i < mx; i++ ) oPoint[i]  = oPoint[i-1] + ny;

for(i=0;i<10;i++)
        for(j=0;j<PARAMS;j++)
                oPoint[i][j] = 0;


fp= fopen("Convergence.txt","w");
fclose(fp);

fp= fopen("ReportDE.html","w");
startHTMLBODY(fp);
for(strategy=1;strategy<=10;strategy++)
{
        printf("Strategy %d\n",strategy);
        Heading1;
        fprintf(fp,"Strategy %d :",strategy);
        endHeading1;
        oResult[strategy-1]       = -1;
        for(NP=NPmin;NP<=NPmax;NP+=sNP)
        {
                printf("NP = %d\n",NP);
                Heading2;
                fprintf(fp,"NP = %d\n",NP);
                endHeading2;
                fprintf(fp,"<TABLE cellpadding=\"0\" cellspacing=\"0\"
                class=\"borderTL\">\n");
                fprintf(fp,"<TR><TD class=\"data1\">NP = %d</TD>\n",NP);
                for(F=Fmin;F<=Fmax;F+=sF)
                fprintf(fp,"<TD class=\"data\" align=\"center\">F = %0.2lf</TD>\n",F);
                fprintf(fp,"</TR>\n");
                for(CR=CRmin;CR<=CRmax;CR+=sCR)
                {
                        fprintf(fp,"<TR><TD class=\"data1\" align=\"center\">CR =
                        %0.2lf</TD>\n",CR);
                        for(F=Fmin;F<=Fmax;F+=sF)
                        {
                                feval = 0;
                                result    = optimise(GENMAX, PARAMS, NP, F, CR,
                                strategy, Point);
                                if(oResult[strategy-1]==-1 || oResult[strategy-1]>result)
                                {
                                        oNFE[strategy-1] = feval;
                                        oResult[strategy-1]       = result;
                                        wResult[strategy-1] = resultpen;
                                        oNP[strategy-1]= NP;
                                        oF[strategy-1]   = F;
                                        oCR[strategy-1]= CR;
```

```
                                        oTime[strategy-1]       = timeTaken;
                                        copyPoint(oPoint[strategy-1],Point,ny);
                                }
                        fprintf(fp,"<TD class=\"data\" align=\"right\">%0.6lE (%ld
                        ms)</TD>\n",result,timeTaken);
                        }
                        fprintf(fp,"</TR>\n");
                }
                fprintf(fp,"</TABLE>\n");
        }
        if(best==0 || oResult[strategy-1] <= oResult[best-1])
                best = strategy;
}

//Minium constraint violation value
resultpen = wResult[0];
for(i=1;i<10;i++) if(wResult[i]<resultpen) resultpen = wResult[i];

//Best objective function value with minimum constraint violation
i = 0;
while(wResult[i]!=resultpen) i++;
result = oResult[i];
best = i+1;
for(i=best;i<10;i++) if(wResult[i]==resultpen && result<oResult[i]) oResult[i]=result;

//Best strategy with time
best = 0;
timeTaken=-1;
for(i=0;i<10;i++)
        if(wResult[i]==resultpen && oResult[i]==result)
                if(timeTaken==-1 || timeTaken > oTime[i])
                {
                        timeTaken = oTime[i];
                        best=i;
                }
best++;

Heading1;
fprintf(fp,"Results : ");
endHeading1;
fprintf(fp,"<TABLE cellpadding=\"0\" cellspacing=\"0\" class=\"borderTL\">\n");
fprintf(fp,"<TR><TD class=\"data1\" align=\"center\"><B>Strategy No.</B></TD>\n");
fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>Strategy</B></TD>\n");
fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>NP</B></TD>\n");
fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>CR</B></TD>\n");
fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>F</B></TD>\n");
fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>Optimal Value</B></TD>\n");
fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>Constraint
Violation</B></TD>\n");
fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>NFE</B></TD>\n");
```

```
	fprintf(fp,"<TD class=\"data1\" align=\"center\"><B>Time
	Taken(ms)</B></TD></TR>\n");
	for(strategy=0;strategy<10;strategy++)
	{
		fprintf(fp,"<TR><TD class=\"data1\">%d</TD>\n",strategy+1);
		fprintf(fp,"<TD class=\"data1\">%s</TD>\n",strategies[strategy]);
		fprintf(fp,"<TD class=\"data1\">%d</TD>\n",oNP[strategy]);
		fprintf(fp,"<TD class=\"data1\">%0.2lf</TD>\n",oCR[strategy]);
		fprintf(fp,"<TD class=\"data1\">%0.2lf</TD>\n",oF[strategy]);
		fprintf(fp,"<TD class=\"data1\">%0.6lE</TD>\n",oResult[strategy]);
		fprintf(fp,"<TD class=\"data1\">%0.4lE</TD>\n",wResult[strategy]);
		fprintf(fp,"<TD class=\"data1\">%d</TD>\n",oNFE[strategy]);
		fprintf(fp,"<TD class=\"data1\">%ld</TD></TR>\n",oTime[strategy]);
	}
	fprintf(fp,"</TABLE>");

	Heading1;
	fprintf(fp,"Best Strategy is Strategy <B>%s</B><BR>",strategies[best-1]);
	endHeading1;

	fprintf(fp,"Minimum constraint violation (CV) : <B>%0.4lE</B><BR>",resultpen);
	fprintf(fp,"Minimum objective value with min CV: <B>%0.6lE</B><BR>",result);
	fprintf(fp,"Minimum time taken : <B>%d</B><BR>",timeTaken);

	fprintf(fp,"<TABLE cellpadding=\"0\" cellspacing=\"0\" class=\"borderTL\">\n");
	fprintf(fp,"<TR>\n");
	fprintf(fp,"<TD class=\"data1\" align=\"center\">Parameter</TD>\n");
	fprintf(fp,"<TD class=\"data1\" align=\"center\">Value</TD></TR>\n");
	for(i=0;i<ny;i++)
	{
		fprintf(fp,"<TR><TD class=\"data1\">X<sub>%d</sub></TD>\n",i+1);
		fprintf(fp,"<TD class=\"data1\" align=\"right\">%lf</TD></TR>\n",oPoint[best-
		1][i]);
	}
	fprintf(fp,"</TABLE>");

	fprintf(fp,"<font size=\"1\">End of report</font>");
	closeHTML(fp);
	fclose(fp);
	return 0;
}

double optimise(int gen_max,int D,int NP,double F,double CR,int strategy,double *Point)
{
	int min_cost, count, a, b, c, d, e, i, j, k, imin;
	double result, cmin, *best, *bestit, *trial, *cost, score, *xlow, *xhigh;
	FILE *fp;
	struct timeb start,end;
	double epsilon;
	count	= 0;
	xlow	= (double *)malloc(sizeof(double)*D);
```

```
xhigh   = (double *)malloc(sizeof(double)*D);
best    = (double *)malloc(sizeof(double)*D);
bestit  = (double *)malloc(sizeof(double)*D);
trial   = (double *)malloc(sizeof(double)*D);
cost    = (double *)malloc(sizeof(double)*NP);

fp      = fopen("limits.txt","r");
for(i=0;i<D;i++)
        fscanf(fp,"%lf %lf",&xlow[i],&xhigh[i]);
fclose(fp);

ftime(&start);

for(i=0;i<NP;i++)
{
        for(j=0;j<D;j++)
                x1[i][j] = xlow[j]+rnd_uni()*(xhigh[j]-xlow[j]);
        cost[i]  = evaluate(x1[i]);
}
cmin = cost[0];
imin = 0;
for (i=1; i<NP; i++)
{
        if (cost[i]<cmin)
        {
                cmin = cost[i];
                imin = i;
        }
}
copyPoint(best,x1[imin],D);       /*save best member ever        */
copyPoint(bestit,x1[imin],D);     /*save best member of generation */

fp      = fopen("Convergence.txt","a");
fprintf(fp,"\n\nStrategy %d (NP=%d; CR=%0.2lf; F=%0.2lf)\n",strategy, NP, CR, F);
fprintf(fp,"-----------------------------------\n");
fclose(fp);

while(!(fabs(epsilon)<EPSILON && count>=gen_max))
{
        for(i=0;i<NP;i++)
        {
                do a=(int)(rnd_uni()*NP); while(a==i);
                do b=(int)(rnd_uni()*NP); while(b==i || b==a);
                do c=(int)(rnd_uni()*NP); while(c==i || c==b || c==a);
                do d=(int)(rnd_uni()*NP); while(d==i || d==c || d==b ||d==a);
                do e=(int)(rnd_uni()*NP); while(e==i || e==d || e==c ||e==b || e==a);

                /*--------------- DE/rand/1/bin ----------------------*/
                if(strategy == 1)
                {
                        j= (int)(rnd_uni()*D);
```

```
		for(k=1;k<=D;k++)
		{
			if(rnd_uni()<CR || k==D)
				trial[j] = x1[c][j]+F*(x1[a][j]-x1[b][j]);
			else
				trial[j] = x1[i][j];

			if(trial[j]>xhigh[j] || trial[j]<xlow[j])
				trial[j]   = x1[i][j];
			j= (j+1)%D;
		}
	}
	/*--------------- DE/best/1/bin ----------------------*/
	else if(strategy == 2)
	{
		j= (int)(rnd_uni()*D);
		for(k=1;k<=D;k++)
		{
			if(rnd_uni()<CR || k==D)
				trial[j] = bestit[j]+F*(x1[a][j]-x1[b][j]);
			else
				trial[j] = x1[i][j];
			if(trial[j]>xhigh[j] || trial[j]<xlow[j])
				trial[j]   = x1[i][j];

			j= (j+1)%D;
		}
	}
	/*--------------- DE/best/2/bin ----------------------*/
	else if(strategy == 3)
	{
		j= (int)(rnd_uni()*D);
		for(k=1;k<=D;k++)
		{
			if(rnd_uni()<CR || k==D)
				trial[j] = bestit[j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-
				x1[d][j]);
			else
				trial[j] = x1[i][j];
			if(trial[j]>xhigh[j] || trial[j]<xlow[j])
				trial[j]   = x1[i][j];
			j= (j+1)%D;
		}
	}

	/*--------------- DE/rand/2/bin ----------------------*/
	else if(strategy == 4)
	{
		j= (int)(rnd_uni()*D);
		for(k=1;k<=D;k++)
		{
```

```
                    if(rnd_uni()<CR || k==D)
                            trial[j] = x1[e][j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-
                            x1[d][j]);
                    else
                            trial[j] = x1[i][j];
                    if(trial[j]>xhigh[j] || trial[j]<xlow[j])
                            trial[j]   = x1[i][j];

                    j= (j+1)%D;
            }
    }

    /*---------- DE/rand-to-best/1/bin ----------*/
    else if(strategy == 5)
    {
            copyPoint(trial,x1[i],D);
            j= (int)(rnd_uni()*D);
            for(k=1;k<=D;k++)
            {
                    if(rnd_uni()<CR || k==D)
                            trial[j] = trial[j]+F*(bestit[j]-
                            trial[j])+F*(x1[a][j]-x1[b][j]);
                    else
                            trial[j] = x1[i][j];
                    if(trial[j]>xhigh[j] || trial[j]<xlow[j])
                            trial[j]   = x1[i][j];
                    j= (j+1)%D;
            }
    }

    /*---------- DE/rand/1/exp ----------*/
    else if(strategy == 6)
    {
            copyPoint(trial,x1[i],D);
            j= (int)(rnd_uni()*D);
            k=0;
            do
            {
                    trial[j] = x1[c][j]+F*(x1[a][j]-x1[b][j]);
                    if(trial[j]>xhigh[j] || trial[j]<xlow[j])
                            trial[j]   = x1[i][j];

                    j= (j+1)%D;
                    k++;
            }while(rnd_uni()<CR && k<D);

    }

    /*---------- DE/best/1/exp ----------*/
    else if(strategy == 7)
    {
```

```
		copyPoint(trial,x1[i],D);
		j= (int)(rnd_uni()*D);
		k=0;
		do
		{
			trial[j] = bestit[j]+F*(x1[a][j]-x1[b][j]);
			if(trial[j]>xhigh[j] || trial[j]<xlow[j])
				trial[j]   = x1[i][j];
			j= (j+1)%D;
			k++;
		}while(rnd_uni()<CR && k<D);

}

/*--------------- DE/best/2/exp ---------------------*/
else if(strategy == 8)
{
		copyPoint(trial,x1[i],D);
		j= (int)(rnd_uni()*D);
		k=0;
		do
		{
			trial[j] = bestit[j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);
			if(trial[j]>xhigh[j] || trial[j]<xlow[j])
				trial[j]   = x1[i][j];
			j= (j+1)%D;
			k++;
		}while(rnd_uni()<CR && k<D);

}

/*--------------- DE/rand/2/exp ---------------------*/
else if(strategy == 9)
{
		copyPoint(trial,x1[i],D);
		j= (int)(rnd_uni()*D);
		k=0;
		do
		{
			trial[j] = x1[e][j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);
			if(trial[j]>xhigh[j] || trial[j]<xlow[j])
				trial[j]   = x1[i][j];
			j= (j+1)%D;
			k++;
		}while(rnd_uni()<CR && k<D);

}

/*--------------- DE/rand-to-best/1/exp ---------------------*/
else if(strategy == 10)
{
```

```
                        copyPoint(trial,x1[i],D);
                        j= (int)(rnd_uni()*D);
                        k=0;
                        do
                        {
                                trial[j] = trial[j]+F*(bestit[j]-trial[j])+F*(x1[a][j]-
                                x1[b][j]);
                                if(trial[j]>xhigh[j] || trial[j]<xlow[j])
                                        trial[j]   = x1[i][j];
                                j= (j+1)%D;
                                k++;
                        } while(rnd_uni()<CR && k<D);

                }

                score    = evaluate(trial);
                if(score<=cost[i])
                {
                        copyPoint(x2[i],trial,D);
                        cost[i]   = score;
                }
                else
                        copyPoint(x2[i],x1[i],D);
        }

        for(i=0;i<NP;i++)
                copyPoint(x1[i],x2[i],D);

        imin = 0;
        for (i=1; i<NP; i++)
        {
        if (cost[i]<cmin)
        {
                cmin = cost[i];
                imin = i;
        }
        }
        count++;
        epsilon = evaluate(bestit);
        fp        = fopen("Convergence.txt","a");
        fprintf(fp,"%d\t%0.6lE\n", count, epsilon);
        fclose(fp);
        copyPoint(bestit,x1[imin],D);       /* save best member of generation */
        epsilon -= evaluate(bestit);                    /* Determines the epsilon value */
}
min_cost          = 0;
for(i=1;i<NP;i++) if(cost[i] < cost[min_cost]) min_cost   = i;
copyPoint(Point,x1[min_cost],D);
result    = evaluateWOpenalty(Point);
resultpen = (fabs(fabs(result)-fabs(evaluate(Point))));
ftime(&end);
```

```
	timeTaken= (end.time-start.time)*1000;
	timeTaken+= (end.millitm-start.millitm);
	return result;
}

double sumarray(double *x,int start,int end)
{
	int i;
	double result=0;
	for(i=start;i<=end;i++)
		result+= x[i];
	return result;
}

int getInt(const char *prompt)
{
	int result;
	printf(prompt);
	scanf("%d",&result);
	return result;
}

double getDouble(const char *prompt)
{
	double result;
	printf(prompt);
	scanf("%lf",&result);
	return result;
}

double rnd_uni()
{
	int r;
	r= rand();
	return (double)(r)/RAND_MAX;
}

void copyPoint(double *dest,double *source,int D)
{
	int i;
	for(i=0;i<D;i++)
		dest[i]   = source[i];
	return;
}

double evaluateWOpenalty(double *x)
{
	double value;
```

```
	/*   DO NOT CHANGE ANYTHING ABOVE   */

	/*----------------------CODE YOUR OBJECTIVE FUNCTIONS HERE------------*/
	/*All functions must be of minimization type*/
	/*Example for Objective Function

	val[0] = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) + (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-
	7);

	-----------------------------------------------------------------------------------------*/

	/*=============Start Coding Your Objective Function==============*/

	value = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) + (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-
	7);

	/*=============END CODING UPTO THIS POINT======================*/
	return value;
}

double evaluate(double *x)
{
	int ncons, i;
	double *val, *penal, temp;
	long double value=0;
	feval++;

	/*   DO NOT CHANGE ANYTHING ABOVE   */
	/* Enter the Number of Constraints to the variable ncons*/

	ncons = 2;

	/*-----------------------------------------------------*/

	val	= (double *)malloc(sizeof(double)*ncons);
	penal	= (double *)malloc(sizeof(double)*ncons);
	penal[0]=1;

	/*----------------------CODE YOUR OBJECTIVE FUNCTIONS HERE------------*/
	/*All functions must be of minimization type*/

	/*Example for Objective Function

	val[0] = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) + (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-
	7);

	-----------------------------------------------------------------------------------------*/
```

```
        /*============Start Coding Your Objective Function=============*/

        val[0] = (x[0]*x[0]+x[1]-11)*(x[0]*x[0]+x[1]-11) + (x[0]+x[1]*x[1]-7)*(x[0]+x[1]*x[1]-
        7);
        /*========END YOUR CODING UPTO THIS POINT============*/
        /*----------------------CODE YOUR CONSTRAINTS HERE------------*/
        /*All functions must be of minimization type, negate maximization functions */
        /*Constraints must be of the following type: g(x) <= 0. Enter the constraint
        equation to temp variable and don't change the expression for val[1].
        Similarly, do the same for other constraints. See the example below.*/

        /*Example for Objective Function

        temp = (x[0]-5)*(x[0]-5) + x[1]*x[1] - 26;
        val[1] = max(temp,0);

        If the constraint of the example is an equality constraint, the constraint should be entered
        as follows:

        val[1] = abs((x[0]-5)*(x[0]-5) + x[1]*x[1] - 26);

        ----------------------------------------------------------------------------------------*/

        /*============Start Coding Your Constraints=============*/

        /* Constraint 1
        temp = (x[0]-5)*(x[0]-5) + x[1]*x[1] - 26;
        val[1] = max(temp,0);

        /* Constraint 2
        temp = 4*x[0] + x[1] - 20;
        val[2] = max(temp,0);

        /*Enter the Penalty Coefficient for Each constraint*/

        penal[1] = 1000;

        penal[2] = 1000;

        /*========END YOUR CODING UPTO THIS POINT============*/

        for (i=0;i<=ncons;i++)
                value = value + penal[i]*val[i];
        return value;
}
```

APPENDIX B
PREVIOUS REPORTS IN THE SERIES

ISSN: (print) 1913-3200; (online) 1913-3219

1. Slobodan P. Simonovic (2001). Assessment of the Impact of Climate Variability and Change on the Reliability, Resiliency and Vulnerability of Complex Flood Protection Systems. Water Resources Research Report no. 038, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 91 pages. ISBN: (print) 978-0-7714-2606-3; (online) 978-0-7714-2607-0.

2. Predrag Prodanovic (2001). Fuzzy Set Ranking Methods and Multiple Expert Decision Making. Water Resources Research Report no. 039, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 68 pages. ISBN: (print) 978-0-7714-2608-7; (online) 978-0-7714-2609-4.

3. Nirupama and Slobodan P. Simonovic (2002). Role of Remote Sensing in Disaster Management. Water Resources Research Report no. 040, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 107 pages. ISBN: (print) 978-0-7714-2610-0; (online) 978-0-7714-2611-7.

4. Taslima Akter and Slobodan P. Simonovic (2002). A General Overview of Multiobjective Multiple-Participant Decision Making for Flood Management. Water Resources Research Report no. 041, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 65 pages. ISBN: (print) 978-0-7714-2612-4; (online) 978-0-7714-2613-1.

5. Nirupama and Slobodan P. Simonovic (2002). A Spatial Fuzzy Compromise Approach for Flood Disaster Management. Water Resources Research Report no. 042, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 138 pages. ISBN: (print) 978-0-7714-2614-8; (online) 978-0-7714-2615-5.

6. K. D. W. Nandalal and Slobodan P. Simonovic (2002). State-of-the-Art Report on Systems Analysis Methods for Resolution of Conflicts in Water Resources Management. Water Resources Research Report no. 043, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London,

Ontario, Canada, 216 pages. ISBN: (print) 978-0-7714-2616-2; (online) 978-0-7714-2617-9.

7. K. D. W. Nandalal and Slobodan P. Simonovic (2003). Conflict Resolution Support System – A Software for the Resolution of Conflicts in Water Resource Management. Water Resources Research Report no. 044, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 144 pages. ISBN: (print) 978-0-7714-2618-6; (online) 978-0-7714-2619-3.

8. Ibrahim El-Baroudy and Slobodan P. Simonovic (2003). New Fuzzy Performance Indices for Reliability Analysis of Water Supply Systems. Water Resources Research Report no. 045, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 90 pages. ISBN: (print) 978-0-7714-2620-9; (online) 978-0-7714-2621-6.

9. Juraj Cunderlik (2003). Hydrologic Model Selection for the CFCAS Project: Assessment of Water Resources Risk and Vulnerability to Changing Climatic Conditions. Water Resources Research Report no. 046, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 40 pages. ISBN: (print) 978-0-7714-2622-3; (online) 978-0-7714-2623-0.

10. Juraj Cunderlik and Slobodan P. Simonovic (2004). Selection of Calibration and Verification Data for the HEC-HMS Hydrologic Model. Water Resources Research Report no. 047, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 29 pages. ISBN: (print) 978-0-7714-2624-7; (online) 978-0-7714-2625-4.

11. Juraj Cunderlik and Slobodan P. Simonovic (2004). Calibration, Verification and Sensitivity Analysis of the HEC-HMS Hydrologic Model. Water Resources Research Report no. 048, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 113 pages. ISBN: (print) 978-0-7714-2626-1; (online) 978-0-7714-2627-8.

12. Predrag Prodanovic and Slobodan P. Simonovic (2004). Generation of Synthetic Design Storms for the Upper Thames River basin. Water Resources Research Report no. 049, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 20 pages. ISBN: (print) 978-0-7714-2628-5; (online) 978-0-7714-2629-2.

13. Ibrahim El-Baroudy and Slobodan P. Simonovic (2005). Application of the Fuzzy Performance Indices to the City of London Water Supply System. Water Resources Research Report no. 050, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 137 pages. ISBN: (print) 978-0-7714-2630-8; (online) 978-0-7714-2631-5.

14. Ibrahim El-Baroudy and Slobodan P. Simonovic (2006). A Decision Support System for Integrated Risk Management. Water Resources Research Report no. 051, Facility

for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 146 pages. ISBN: (print) 978-0-7714-2632-2; (online) 978-0-7714-2633-9.

15. Predrag Prodanovic and Slobodan P. Simonovic (2006). Inverse Flood Risk Modelling of The Upper Thames River Basin. Water Resources Research Report no. 052, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 163 pages. ISBN: (print) 978-0-7714-2634-6; (online) 978-0-7714-2635-3.

16. Predrag Prodanovic and Slobodan P. Simonovic (2006). Inverse Drought Risk Modelling of The Upper Thames River Basin. Water Resources Research Report no. 053, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 252 pages. ISBN: (print) 978-0-7714-2636-0; (online) 978-0-7714-2637-7.

17. Predrag Prodanovic and Slobodan P. Simonovic (2007). Dynamic Feedback Coupling of Continuous Hydrologic and Socio-Economic Model Components of the Upper Thames River Basin. Water Resources Research Report no. 054, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 437 pages. ISBN: (print) 978-0-7714-2638-4; (online) 978-0-7714-2639-1.

18. Subhankar Karmakar and Slobodan P. Simonovic (2007). Flood Frequency Analysis Using Copula with Mixed Marginal Distributions. Water Resources Research Report no. 055, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 144 pages. ISBN: (print) 978-0-7714-2658-2; (online) 978-0-7714-2659-9.

19. Jordan Black, Subhankar Karmakar and Slobodan P. Simonovic (2007). A Web-Based Flood Information System. Water Resources Research Report no. 056, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 133 pages. ISBN: (print) 978-0-7714-2660-5; (online) 978-0-7714-2661-2.

20. Angela Peck, Subhankar Karmakar and Slobodan P. Simonovic (2007). Physical, Economical, Infrastructural and Social Flood Risk – Vulnerability Analyses in GIS. Water Resources Research Report no. 057, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 80 pages. ISBN: (print) 978-0-7714-2662-9; (online) 978-0-7714-2663-6.

21. Predrag Prodanovic and Slobodan P. Simonovic (2007). Development of Rainfall Intensity Duration Frequency Curves for the City of London Under the Changing Climate. Water Resources Research Report no. 058, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 51 pages. ISBN: (print) 978-0-7714-2667-4; (online) 978-0-7714-2668-1.

22. Evan G. R. Davies and Slobodan P. Simonovic (2008). An integrated system dynamics model for analyzing behaviour of the social-economic-climatic system: Model description and model use guide. Water Resources Research Report no. 059, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 233 pages. ISBN: (print) 978-0-7714-2679-7; (online) 978-0-7714-2680-3.