

# **Evolutionary Computation for Optimization of Selected Non-Linear Chemical Processes**

**THESIS**

**Submitted in partial fulfillment  
of the requirements for the degree of  
DOCTOR OF PHILOSOPHY**

**By**

**RAKESH ANGIRA**

**Under the supervision of**

**Prof. (Dr.) B. V. BABU**



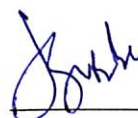
**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE  
PILANI (RAJASTHAN) INDIA  
2005**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE  
PILANI (RAJASTHAN) INDIA**

**CERTIFICATE**

This is to certify that the thesis entitled "Evolutionary Computation for Optimization of Selected Non-Linear Chemical Processes" and submitted by Rakesh Angira ID. No. 2000PHXF011 for award of Ph.D. Degree of the Institute, embodies the original work done by him under my supervision.

Signature in full of the Supervisor



Name in capital block letters

B. V. BABU

Date: 18-10-2005

**Designation**

Professor of Chemical Engineering  
Group Leader - Chemical Engineering &  
Assistant Dean - Engineering Services Division

To  
My Father

---

## ACKNOWLEDGEMENTS

---

I wish to express deep sense of gratitude and sincere thanks to my guide Prof. (Dr.) B. V. Babu for allowing me to work in the emerging area of Evolutionary Computation, helping me understand the concepts, continuous encouragement, providing me all guidance and moral support. It is from him that I learned the traits of research and developed a desire to excel.

I thank Prof. S. Venkateswaran, Vice-Chancellor, BITS, Pilani for allowing me to pursue my thesis in the Institute.

My sincere thanks to Prof. L. K. Maheshwari – Pro Vice-Chancellor, Prof. K. E. Raman - Deputy Director (Administration), Prof. V. S. Rao - Deputy Director (Off-Campus Programmes), Prof. Ravi Prakash – Dean (R & C Division), and Prof. A. K. Sarkar - Dean (Instruction Division and Faculty Division I), for providing the necessary infrastructure and other facilities whenever required.

I also express my gratitude for the kind and affectionate enquiries about the work, providing the computing facility & the encouragement given by Prof. B. R. Natrajan – Dean (DLP).

Special thanks and appreciation is extended to Prof. R. P. Vaid, Prof. T. N. S. Mathur, Dr. S. D. Manjare, Mr. B. D. Munshi, Mr. H. K. Mohanta, and other faculty members of Chemical Engineering Group for their valuable advice and moral support throughout the study.

Special thanks are also extended to Mr. Pankaj Vyas, Dr. Deepak Satpati, Dr. N. N. Sharma, Dr. B. K. Mishra, Dr. S. D. Pohekar, and Dr. V. Ramakrishna for their cooperation and encouragement in completing my research work.

I sincerely thank anonymous reviewers of International Journals of 'Computers & Chemical Engineering' for accepting research paper for publication based on the present study.

Sincere thanks to Dr. M. Ishwara Bhat, Librarian, BITS as well as to librarian at IIT, Delhi for providing literature useful for the present study. Special thank to my friend Mr. Ravindra Marathe - Research scholar (National University of Singapore) for sending various research papers throughout the study. I thank my co-researchers Mr. B. K. Rout, Mr. K. K. Gupta, Mr. R. R. Singh, and Mr. Sunil Kumar for their cooperation and encouragement in completing my research work.

I express my thanks to Mr. Narender Saini, Mr. Ashok Jitawat, and other members of DLP Division for their help and cooperation. I also wish to acknowledge Mr. Babulal Saini, Mr. Jangvir, Mr. Shankar, Mr. Jeevan, Mr. Ashok and other non-teaching staff of Chemical Engineering Group for their help and cooperation. I express my thanks to Mr. V. N. Sharma, Mr. Mahendra Saini and Mr. Rajkumar Saini of ESD (Workshop) for their help while sending research papers related to thesis work. I would like to thank members of Reprography, Xerox and Printing Sections for their prompt services.

I would be failing in my duties if I do not mention the name of my wife – Priyanka, daughters - Riya & Vrinda for their constant help and encouragement without disturbing me while spending nights at work. This work would not have been completed without the moral support I got from my brothers – Mr. Rajesh Angira and Mr. Mahesh Angira and loving parents – Mrs. Tripta Angira and Late Dr. O. P. Angira.

Last but not the least, I would like to thank one and all who have helped me in myriad ways throughout the course of this work.

  
Rakesh Angira

---

## ABSTRACT

---

The chemical industry has undergone significant changes during the past 25 years due to the increased cost of energy and increasingly stringent environmental regulations. Most observers both from academia and industry believe that the emphasis in the near future will be on improving efficiency and increasing profitability of existing plants rather than on plant expansion. One of the most important engineering tools that can be employed in such activities is 'optimization'. Unfortunately, none of the traditional algorithms are guaranteed to find the global optimal solution but population based search algorithms are found to have a better global perspective than the traditional methods.

Now-a-days, non-traditional search and optimization methods (also called Evolutionary Computation Methods) that often rely on physical analogies in order to generate trial points that mimic the approach to an equilibrium condition, are becoming popular. Examples include Simulated Annealing (SA), Genetic Algorithms (GA) and Differential Evolution (DE) to name a few. DE, a recent optimization technique, is an exceptionally simple evolution strategy, which is significantly faster & robust at numerical optimization and is more likely to find a function's true global optimum. But the application of DE to chemical processes is scarce.

The present thesis deals with the application of Differential Evolution, an evolutionary computation technique, for the optimization of selected nonlinear chemical processes. There are many chemical processes which are highly nonlinear and complex with reference to optimal operating conditions with many equality and inequality constraints; such as optimization of a thermal cracker, optimal operation of alkylation unit, optimal design of ammonia reactor, reactor network design, and optimal design of heat exchanger network, multi-product batch plant design, dynamic optimization of a batch reactor etc. These processes represent various types of optimization problems such as linear programming problems, non-linear programming problems, process synthesis and design (mixed integer non-linear programming) problems, and dynamic optimization problems etc.

The *performance of DE is evaluated* using various benchmark test functions and selected nonlinear chemical processes. Also the performance of DE, for a given problem, is compared with that of other traditional/evolutionary optimization methods. It is important to note that for some of the problems DE is able to locate better solution (possibly global optimum) than that reported in literature. In case of optimal design of ammonia synthesis reactor a possible error in the NAG routine is found along with the new value of optimum reactor length, and profit function. DE takes large computational time for problems involving computationally expensive

objective functions. In order to reduce the computational efforts, an attempt is made to seek improvements on original DE by finding better mutation and crossover schemes or by hybridizing it with traditional methods.

A *Modified Differential Evolution* (MDE) is proposed and its performance is compared with that of DE and other optimization methods with which the given problem has been solved in literature. Two methods for handling bound violations are studied and compared. Also, a new approach for handling binary variable is proposed and compared with nonlinear transformation modeling for binary variables. The proposed approach for handling binary variables is found to be better than the latter. Further the performance of MDE is found to be better than that of DE and some other methods.

A new *Hybrid DE* (HDE) algorithm is proposed and discussed. This new method is a hybrid of DE and quasi-Newton method. HDE located the global optimum with high accuracy as compared to DE, Enhanced Continuous Tabu Search, a hybrid method of Tabu Search & Quasi-Newton, and GA. The HDE took less CPU-time as compared to DE. Further, the performance of HDE algorithm is evaluated by its application to various chemical engineering problems and comparing its performance with that of DE.

Also, *two new strategies of DE*, viz., New Strategy-1 (NS-1) and New Strategy-2 (NS-2) are proposed and tuned to their best possible control parameters. In the proposed strategies, the best features of the three existing DE strategies are exploited in order to increase the convergence rate while maintaining the robustness of the algorithm. Further they are applied to selected difficult nonlinear chemical engineering problems and their performance is compared with that of DE and MDE. The proposed strategies are found to be competitive with DE and MDE algorithms.

Two novel techniques to solve multi-objective optimization problems, *Non-dominated Sorting Differential Evolution* (NSDE) and *Modified Non-dominated Sorting Differential Evolution* (MNSDE) are proposed and described in detail with the help of two benchmark test problems. Further the effect of control parameters (*CR* – crossover constant & *F* – scaling factor) on the performance of NSDE and MNSDE is investigated. It is recommended to use a high *CR* value for both NSDE & MNSDE and lower value of *F* for MNSDE and a value of 0.5 for NSDE. The maximum number of generations (*Max\_gen*) is found to be problem dependent.

The proposed new evolutionary algorithms (MDE and HDE), two new strategies of DE (NS-1 & NS-2), and new evolutionary multi-objective optimization algorithms (NSDE and MNSDE) are very useful in solving highly complex real world single and multi-objective optimization problems. And all these newly developed algorithms in this study are found to outperform the existing evolutionary algorithms.

**Keywords:** *Optimization; Evolutionary Algorithms, Differential Evolution; Genetic Algorithms; Chemical processes; Ammonia synthesis reactor; Alkylation; Thermal cracking; Liquid extraction; Heat exchanger network design; Reactor network design; Drying; Simulation; Modified Differential Evolution; Dynamic optimization; Process synthesis and design; Nonlinear programming problems; Mixed integer nonlinear programming problems.*

# TABLE OF CONTENTS

Acknowledgments	i
Abstract	iii
Table of contents	v
List of figures	x
List of tables	xiv
List of symbols	xvii
<b>1. Introduction</b>	<b>1</b>
1.1 Optimization	2
1.2 Traditional Methods & their limitations	4
1.3 Evolutionary Computation (EC)	6
1.3.1 History	6
1.3.2 What is EC?	6
1.3.2.1 Paradigms of EC	8
1.3.2.1.1 Evolutionary Programming	9
1.3.2.1.2 Evolution Strategies	10
1.3.2.1.3 Genetic Algorithms	11
1.3.3 Why EC?	13
1.3.3.1 Optimization	13
1.3.3.2 Robust adaptation	13
1.3.3.3 Machine intelligence	14
1.3.3.4 Understanding of biology	14
1.4 EC vs. Traditional Methods	14
1.5 Motivation	15
1.6 Objectives	18
1.7 Scope of the present study	19
1.8 Structure of the thesis	20
<b>2. Evolutionary Computation Methods</b>	<b>22</b>
2.1 Genetic Algorithm (GA)	23
2.1.1 Stepwise Procedure of GA	24
2.1.2 Applications of GA	24
2.2 Differential Evolution (DE)	25
2.2.1 Working of DE	29
2.2.2 Applications of DE	32
2.3 Comparison of GA and DE	33
2.4 Application of DE to a Test function	33
2.5 Conclusions	36



3.5.6.2	Problem Formulation	100
3.5.6.3	Results and Discussion	101
3.5.6.4	Conclusions	103
3.5.7	Optimum Fuel Allocation in Power Plant (FAPP)	104
3.5.7.1	Introduction	104
3.5.7.2	The Problem	105
3.5.7.2.1	Problem Formulation	106
3.5.7.3	Results and Discussion	107
3.5.7.4	Conclusions	109
3.5.8	Water Pumping System	110
3.5.8.1	Introduction	110
3.5.8.2	The Problem	110
3.5.8.2.1	Problem Modification	112
3.5.8.2.2	Problem Reformulation-1	112
3.5.8.3	Results and Discussion	113
3.5.8.3.1	Problem Reformulation-2	115
3.5.8.4	Conclusions	116
3.5.9	Liquid Extraction Problem	116
3.5.9.1	Introduction	116
3.5.9.2	Process Model	118
3.5.9.2.1	Objective Function	121
3.5.9.3	Results and Discussion	121
3.5.9.4	Conclusions	122
3.5.10	Heat Exchanger Network Design (HEND)	123
3.5.10.1	Introduction	123
3.5.10.2	Problem Formulation	123
3.5.10.3	Results and Discussion	125
3.5.10.4	Conclusions	127
<b>4.</b>	<b>Application of MDE to Process Synthesis and Design</b>	<b>128</b>
4.1	Introduction	128
4.2	Background	129
4.3	Problem Formulation	132
4.4	Handling of Integer and Binary variables	133
4.4.1	Integer variables	133
4.4.2	Binary or discrete variables	134
4.4.2.1	Approach-1	134
4.4.2.2	Approach-2	134
4.5	Test Problems on Process Synthesis and Design	135
4.6	Results and Discussion	142
4.6.1	Approach-1	143
4.6.1.1	Comparison of DE and MDE	144
4.6.2	Approach-2	146
4.6.2.1	Comparison of DE and MDE	148
4.6.3	Comparison of Approach-1 and Approach-2	150
4.6.4	Comparison of MDE, GA, M-SIMPSA, and ES ( $\mu + \lambda$ )	156
4.7	Conclusions	159
<b>5.</b>	<b>HDE and its Application to Test functions and Selected Nonlinear Chemical Processes</b>	<b>160</b>
5.1	Introduction	160

5.2 HDE algorithm	161
5.3 Test Functions	163
5.3.1 Results and Discussion	163
5.4 Selected Non-linear Chemical Processes	167
5.4.1 Water Pumping System	168
5.4.1.1 Results and Discussion	168
5.4.2. Heat Exchanger Network Design (HEND)	170
5.4.2.1. Problem Reformulation	170
5.4.2.2. Results and Discussion	171
5.4.3. Three Stage Compressor with Inter-cooling	172
5.4.3.1. Problem Formulation	173
5.4.3.2. Results and Discussion	174
5.4.4. Drying Problem	175
5.4.4.1. Results and Discussion	175
5.5 Conclusions	176
<b>6. New Strategies of DE and their Application to Selected Non-Linear Chemical Processes</b>	<b>178</b>
6.1 Introduction	178
6.2 New Strategies of DE	179
6.3 Test Functions	181
6.3.1 Results and Discussion	182
6.3.1.1 New Strategy-1	182
6.3.1.2 New Strategy-2	186
6.4 Selected Non-linear Chemical Processes	189
6.4.1 Multi-Product Batch Plant	189
6.4.1.1 Results and Discussion	189
6.4.2 Reactor Network Design	191
6.4.2.1 Results and Discussion	191
6.4.3 Dynamic optimization of a Batch Reactor	193
6.4.3.1 Results and Discussion	194
6.5 Conclusions	196
<b>7. An Extension of DE and MDE for Multi-Objective Optimization</b>	<b>197</b>
7.1 Introduction	197
7.2 Multi Objective Optimization Problems (MOOPs)	199
7.2.1 Pareto Optimal Solutions	200
7.3 Non-dominated Sorting Differential Evolution (NSDE)	201
7.3.1 Test Problems	203
7.3.2 Results and Discussion	204
7.3.2.1 Schaffer's Function	204
7.3.2.1.1 Effect of <i>Max_gen</i>	204
7.3.2.1.2 Effect of <i>CR</i> and <i>F</i>	206
7.3.2.2 Cantilever design problem	209
7.3.2.2.1 Effect of <i>Max_gen</i>	209
7.3.2.2.2 Effect of <i>CR</i> and <i>F</i>	210
7.3.3 Conclusions	214
7.4 Modified Non-dominated Sorting Differential Evolution (MNSDE)	215
7.4.1 Results and Discussion	216
7.4.1.1 Schaffer's Function	216

7.4.1.1.1	Effect of <i>Max_gen</i> on MNSDE & its Comparison with NSDE	217
7.4.1.1.2	Effect of <i>CR</i> on MNSDE & its Comparison with NSDE	217
7.4.1.1.3	Effect of <i>F</i> on MNSDE & its Comparison with NSDE	220
7.4.1.2	Cantilever design problem	221
7.4.1.2.1	Effect of <i>Max_gen</i> on MNSDE & its Comparison with NSDE	221
7.4.1.2.2	Effect of <i>F</i> on MNSDE & its Comparison with NSDE	222
7.4.1.2.3	Effect of <i>CR</i> on MNSDE & its Comparison with NSDE	223
7.4.2	Conclusions	226
7.5	Overall Conclusions	226
7.5.1	Shaffer's Function	226
7.5.2	Cantilever Design Problem	227
<b>8.</b>	<b>Concluding Remarks</b>	228
8.1	Summary	228
8.1.1	Introduction	228
8.1.2	Comparison of DE with GA	229
8.1.3	Modified Differential Evolution (MDE)	230
8.1.4	Hybrid Differential Evolution (HDE)	231
8.1.5	New Strategies of DE	231
8.1.6	Extension of DE and MDE for MOOPs	232
8.2	Conclusions	233
8.3	Major Contributions	234
8.4	Future Scope for Research	235
	<b>References</b>	236
	<b>List of Publications</b>	252
	<b>Biographies</b>	254
	<b>Appendix I</b>	256
	<b>Appendix II</b>	262
	<b>Appendix III</b>	290
	<b>Appendix IV</b>	293
	<b>Appendix V</b>	295
	<b>Appendix VI</b>	303

# LIST OF FIGURES

1.1	Types of optimization problems	3
1.2	A typical evolutionary algorithm	8
1.3	The evolutionary programming algorithm	9
1.4	The evolution strategy algorithm	11
1.5	The genetic algorithm	12
2.1	Schematic of working of DE	28
2.2	Comparison of DE and GA	36
3.1	Schematic of DE	40
3.2	Schematic of MDE	40
3.3	Effect of $CR$	43
3.4	Effect of $F$	43
3.5	Error variation for $GP_2$	48
3.6	Error variation for $H_3$	48
3.7	Error variation for $ES_2$	49
3.8	Error variation for $Z_{10}$ ((2 to 10) x 5 generations)	49
3.9	Error variation for $Z_{10}$ ((11 to 250) x 5 generations)	50
3.10	Error variation for $R_{10}$	50
3.11	Classification of cracking process	53
3.12	Thermal cracker	55
3.13	Variation of objective function value with ethylene processing constraint	62
3.14	Auto-thermal ammonia synthesis reactor	67
3.15	Energy and material balance on control volume	70
3.16	Computation procedure	72
3.17	Profile obtained using RKFS (step size = 0.01)	73
3.18	Profile obtained using EULER (step size = 0.01)	74
3.19	Profile obtained using POLYMATH (step size = 0.01)	74
3.20	Profile obtained using D02EJF (step size = 0.01)	75
3.21	Profile obtained using D02EJF (step size = 0.001)	75

3.22	Profile obtained using RKFS for top temperature of 664 K	77
3.23	Profile obtained using RKFS for top temperature of 640 K	78
3.24	Profile obtained using RKFS for top temperature of 600 K	78
3.25	Profile obtained using D02EJF for top temperature of 664 K	79
3.26	Variation of objective function with reactor length (various numerical methods)	81
3.27	Magnification of highlighted part of Fig. 3.26	81
3.28	Objective function variation with reactor length at various top temperatures using RKFS	82
3.29	The variation of optimum reactor length with top temperature	82
3.30	Schematic of reactor network design problem	86
3.31	Convergence history of RND problem	89
3.32	Variation of $f$ with time and temperature (temperature range: 500-1000K)	93
3.33	Convergence history of isothermal CSTR design problem	93
3.34	Simplified alkylation process flow sheet	95
3.35	Convergence history of alkylation problem	100
3.36	Convergence history of drying process problem	104
3.37	Schematic of two-boiler-turbine-generator power plant	105
3.38	Convergence history of FAPP problem	109
3.39	Schematic of water pumping system	111
3.40	Schematic of extraction column	119
3.41	Schematic of heat exchanger network design problem	124
3.42	Convergence history of HEND problem	126
4.1	Superstructure for two-reactor problem	139
4.2	$NFE$ variation using DE and MDE (MWFB, Approach-1)	145
4.3	$NRC$ variation using DE and MDE (MWFB, Approach-1)	145
4.4	$NFE$ variation using DE and MDE (FBM, Approach-1)	147
4.5	$NRC$ variation using DE and MDE (FBM, Approach-1)	147
4.6	$NFE$ variation using DE and MDE (MWFB, Approach-2)	149
4.7	$NRC$ variation using DE and MDE (MWFB, Approach-2)	149
4.8	$NFE$ variation using DE and MDE (FBM, Approach-2)	151
4.9	$NRC$ variation using DE and MDE (FBM, Approach-2)	151
4.10	$NFE$ variation using DE for Approach-1 and Approach-2 (MWFB)	152
4.11	$NRC$ variation using DE for Approach-1 and Approach-2 (MWFB)	152

4.12	<i>NFE</i> variation using MDE for Approach-1 and Approach-2 (MWFB)	154
4.13	<i>NRC</i> variation using MDE for Approach-1 and Approach-2 (MWFB)	154
4.14	<i>NFE</i> variation using DE for Approach-1 and Approach-2 (FBM)	155
4.15	<i>NRC</i> variation using DE for Approach-1 and Approach-2 (FBM)	155
4.16	<i>NFE</i> variation using MDE for Approach-1 and Approach-2 (FBM)	157
4.17	<i>NRC</i> variation using MDE for Approach-1 and Approach-2 (FBM)	157
5.1	AD variation for GP <sub>2</sub> function using DE and HDE	166
5.2	AD variation for WPS problem using DE and HDE	169
5.3	AD variation for HEND problem using DE and HDE	172
5.4	Three-stage compressor with inter-cooling	172
5.5	AD variation for MWC problem using DE and HDE	174
5.6	AD variation for drying problem using DE and HDE	176
6.1	Effect of <i>CR</i> and <i>F</i> on Ackley's function using NS-1	183
6.2	Effect of <i>CR</i> and <i>F</i> on Rastrigin's function using NS-1	183
6.3	Convergence history for Ackley's function using DE and NS-1	185
6.4	Convergence history for Rastrigin's function using DE and NS-1	185
6.5	Effect of <i>CR</i> and <i>F</i> on Ackley's function using NS-2	186
6.6	Effect of <i>CR</i> and <i>F</i> on Rastrigin's function using NS-2	187
6.7	Convergence history for Ackley's function using DE and NS-2	188
6.8	Convergence history for Rastrigin's function using DE and NS-2	188
6.9	Convergence history of RND problem	192
6.10	Optimal temperature profile	195
7.1	Pareto optimal front for Schaffer's function ( <i>NP</i> = 40)	205
7.2	Effect of <i>Max_gen</i> on <i>NPS</i>	205
7.3	Pareto optimal front for Schaffer's function ( <i>NP</i> = 400)	206
7.4	Pareto optimal front for Schaffer's function for different seed values	207
7.5	Pareto optimal front for Schaffer's function for different <i>F</i> values	208
7.6	Pareto optimal front for cantilever design problem	209
7.7	Effect of <i>Max_gen</i> on <i>NPS</i> (cantilever design problem)	210
7.8	Pareto optimal front for different <i>F</i> values (cantilever design problem)	211
7.9	Effect of scaling factor ( <i>F</i> ) on <i>NPS</i> (cantilever design problem)	212
7.10	Pareto optimal fronts for different <i>CR</i> values (cantilever design problem)	213
7.11	Effect of <i>CR</i> on <i>NPS</i> (cantilever design problem)	214
7.12	Pareto front using MNSDE and NSDE (Schaffer's function)	216

4.12	<i>NFE</i> variation using MDE for Approach-1 and Approach-2 (MWFB)	154
4.13	<i>NRC</i> variation using MDE for Approach-1 and Approach-2 (MWFB)	154
4.14	<i>NFE</i> variation using DE for Approach-1 and Approach-2 (FBM)	155
4.15	<i>NRC</i> variation using DE for Approach-1 and Approach-2 (FBM)	155
4.16	<i>NFE</i> variation using MDE for Approach-1 and Approach-2 (FBM)	157
4.17	<i>NRC</i> variation using MDE for Approach-1 and Approach-2 (FBM)	157
5.1	AD variation for GP <sub>2</sub> function using DE and HDE	166
5.2	AD variation for WPS problem using DE and HDE	169
5.3	AD variation for HEND problem using DE and HDE	172
5.4	Three-stage compressor with inter-cooling	172
5.5	AD variation for MWC problem using DE and HDE	174
5.6	AD variation for drying problem using DE and HDE	176
6.1	Effect of <i>CR</i> and <i>F</i> on Ackley's function using NS-1	183
6.2	Effect of <i>CR</i> and <i>F</i> on Rastrigin's function using NS-1	183
6.3	Convergence history for Ackley's function using DE and NS-1	185
6.4	Convergence history for Rastrigin's function using DE and NS-1	185
6.5	Effect of <i>CR</i> and <i>F</i> on Ackley's function using NS-2	186
6.6	Effect of <i>CR</i> and <i>F</i> on Rastrigin's function using NS-2	187
6.7	Convergence history for Ackley's function using DE and NS-2	188
6.8	Convergence history for Rastrigin's function using DE and NS-2	188
6.9	Convergence history of RND problem	192
6.10	Optimal temperature profile	195
7.1	Pareto optimal front for Schaffer's function ( <i>NP</i> = 40)	205
7.2	Effect of <i>Max_gen</i> on <i>NPS</i>	205
7.3	Pareto optimal front for Schaffer's function ( <i>NP</i> = 400)	206
7.4	Pareto optimal front for Schaffer's function for different seed values	207
7.5	Pareto optimal front for Schaffer's function for different <i>F</i> values	208
7.6	Pareto optimal front for cantilever design problem	209
7.7	Effect of <i>Max_gen</i> on <i>NPS</i> (cantilever design problem)	210
7.8	Pareto optimal front for different <i>F</i> values (cantilever design problem)	211
7.9	Effect of scaling factor ( <i>F</i> ) on <i>NPS</i> (cantilever design problem)	212
7.10	Pareto optimal fronts for different <i>CR</i> values (cantilever design problem)	213
7.11	Effect of <i>CR</i> on <i>NPS</i> (cantilever design problem)	214
7.12	Pareto front using MNSDE and NSDE (Schaffer's function)	216

7.13	Effect of <i>Max_gen</i> on <i>NPS</i> using MNSDE and NSDE (Schaffer's function)	217
7.14	Effect of <i>CR</i> using MNSDE (Schaffer's function)	218
7.15	Effect of seed for Schaffer's function	219
7.16	Effect of <i>F</i> on <i>NPS</i> for Schaffer's function (MNSDE and NSDE)	220
7.17	Comparison of MNSDE and NSDE at $F = 0.2$	220
7.18	Effect of <i>Max_gen</i> on <i>NPS</i> for cantilever design problem (MNSDE and NSDE)	221
7.19	Effect of <i>F</i> on <i>NPS</i> for cantilever design problem using MNSDE and NSDE ( $CR = 0.5$ )	222
7.20	Pareto front for cantilever design problem (MNSDE and NSDE)	223
7.21	Effect of <i>CR</i> on <i>NPS</i> for cantilever design problem (MNSDE and NSDE)	224
7.22	Effect of <i>CR</i> on Pareto front for cantilever design problem (MNSDE)	225



# LIST OF TABLES

2.1	Initial Population	29
2.2	Mutation and Crossover	31
2.3	Next Generation Population	31
2.4	Comparison of GA and DE	33
2.5	Results of GA	34
2.6	Results of DE with all ten strategies	34
2.7	Results of Strategy DE/best/1/bin	35
2.8	Results of Strategy DE/best/1/exp	35
3.1	Details of global minimum	46
3.2	Results of DE and MDE	47
3.3	Product Matrix with yield structure: (weight fraction)	56
3.4	Fuel requirements for each feedstock type	56
3.5	Heating value of fuels	56
3.6	Cost of feed, product and fuel (cents/lb)	56
3.7	Results of LP and Simplex and DE	62
3.8	Results of DE (all ten strategies)	63
3.9	Results of MDE and DE for Thermal cracker problem	64
3.10	Comparison of different numerical methods at $x = 10m$	76
3.11	Reactor length at which variables $N_{N_2}$ & $T_g$ intersect	76
3.12	Reactor length at which variables $N_{N_2}$ & $T_f$ intersect	76
3.13	Optimal Length and Profit obtained from various methods	83
3.14	Comparison with literature values	83
3.15	Results of MDE <sup>#</sup> and its comparison with DE <sup>#</sup> and $\alpha$ BB algorithms	88
3.16	Values of $C_i$ and $E_i$	90
3.17	Results reported in literature	91
3.18	Computational Results using DE and MDE for Isothermal Reactor	92
3.19	Variables and their bounds	96

3.20	Comparison of Various Methods	98
3.21	Results of DE and MDE for Alkylation problem	99
3.22	Comparison of DE with DSM and CRS methods (DP problem)	102
3.23	Results of MDE and DE algorithms for DP problem	103
3.24	Comparison of DE with DSM and CRS methods (FAPP problem)	108
3.25	Results of MDE and DE algorithms for FAPP problem	108
3.26	Comparison of DE with Branch & Reduce method	114
3.27	Results of DE with all ten strategies (accuracy = $10^{-6}$ )	114
3.28	Results of DE with all ten strategies (accuracy = $10^{-7}$ )	114
3.29	Results of DE and MDE using MWFB	115
3.30	Results of DE and MDE using FBM	115
3.31	Results of DE and MDE for liquid extraction problem (MWFB)	122
3.32	Results of DE and MDE for liquid extraction problem (FBM)	122
3.33	Comparison of DE with MDE and $\alpha$ BB algorithm (HEND problem)	125
4.1	Characteristics of the test problems	136
4.2	Constants for Problem-6	141
4.3	Values of $S_{ij}$ and $t_{ij}$ of Problem-7	142
4.4	Results of DE using Approach-1	143
4.5	Results of MDE using Approach-1	144
4.6	Results of DE using Approach-2	146
4.7	Results of MDE using Approach-1	148
4.8	Comparison of MDE, GA, ES, M-SIMPSA & M-SIMPSA-pen	158
5.1	Comparison of DE with HDE	164
5.2	Comparison of AAD for various algorithms	166
5.3	Comparison of Success Rate (SR) for various algorithms	166
5.4	Comparison of DE with HDE for WPS problem	169
5.5	Comparison of DE with HDE for HEND problem	171
5.6	Comparison of DE with HDE for MWC problem	174
5.7	Comparison of DE with HDE for drying problem	176
6.1	Control parameters for two test functions	184
6.2	Comparison of NS-1 and NS-2 MPBP problem (FBM)	189
6.3	Comparison of NS-1 and NS-2 MPBP problem at higher CR (FBM)	190
6.4	Comparison of NS-1 and NS-2 MPBP problem (MWFB)	190

6.5	Comparison of NS-1, NS-2, DE, and MDE for RND problem	191
6.6	Comparison of NS-1, DE, and MDE for RND problem	193
6.7	Comparison of NS-1, NS-2, and DE for Batch Reactor problem	194
7.1	Variation of objective function values (min and max) with $F$	212
7.2	Variation of objective function values (min and max) with $CR$	214
7.3	Comparison of maximum function values for two different seeds	218
7.4	Effect of $CR$ on maximum value of objective functions	226

$m$	Distribution coefficient ( $m = 1.5$ )
$Max\_gen$	Maximum number of generations allowed in case of MOOPs
$N/NP$	Population size
$NFE$	Mean number of objective function evaluations
$N_{ox}$	Number of transfer units
$NRC$	Percentage of convergencies to the global optimum
$P(G)$	Population at $G^{th}$ generation
$p_c$	Crossover probability in GA
$R$	Penalty in case of constraint violation/Gas constant
$p_m$	Mutation probability in GA
$S_1$	Surface area of cooling tubes per unit length of reactor (m)
$S_2$	Cross-sectional area of catalyst zone ( $m^2$ )
$T_0$	Top temperature (K)
$T_g$	Temperature of Reacting gas (K)
$U$	Overall heat transfer coefficient ( $kCal/m^2.hr.K$ )
$W$	Total mass flow rate (kg/hr)
$X$	Dimensionless raffinate phase concentration
$x$	Distance along axis (m)
$Y$	Dimensionless extract phase concentration
$Z$	Dimensionless contactor length
$OF_{cal}$	Objective function value at the best point found in each successful experiment
$OF_{Anal}$	Known global minimum
<i>Greek symbols</i>	
$\Delta H$	Heat of reaction ( $kCal/kg$ mole of $N_2$ )
$\varepsilon$	Accuracy
$\mu$	Number of parents at a generation
$\lambda$	Number of offspring at a generation
$ES (\mu + \lambda)$	Selection takes place between the $(\mu + \lambda)$ members
<i>Subscripts</i>	
0	Initial

$i, j$	$i$ th and $j$ th vector/individual
a, b, c	Randomly selected vector/individual
count/G	Generation counter
A, B	Component/Material
min	Minimum
max	Maximum
$D$	Dimension of the problem
$f$	Feed gas
$g$	Reacting gas
$N_2$	Nitrogen
<i>Superscripts</i>	
lo	Lower limit of a variable
hi	Upper limit of a variable
$j$	$j$ th vector/individual
$k$	$k$ -th generation
$r_1, r_2, r_3$	Randomly selected vector/individual
<i>Abbreviations</i>	
AD	Absolute Deviation ( $ OF_{cal} - OF_{Anal} $ )
AAD	Average Absolute Deviation
ACO	Ant Colony Optimization
DE	Differential Evolution
DP	Drying Process
DSM	Direct Search Method
CRS	Controlled Random Search
FAPP	Fuel Allocation in Power Plant
EAs	Evolutionary Algorithms
EC	Evolutionary Computation
ECTS	Enhanced Continuous Tabu Search
EMO	Evolutionary Multi-objective Optimization
EP	Evolutionary Programming
ES	Evolution Strategy
ES <sub>2</sub>	Easom function
FBM	Forced Bound Method

GA	Genetic Algorithm
GP <sub>2</sub>	Goldstein and Price function
H <sub>3</sub>	Hartmann function
HDE	Hybrid Differential Evolution
HEND	Heat Exchanger Network Design
HIM	Himmelblau function
LP	Linear Programming
MA	Memetic Algorithm
MDE	Modified Differential Evolution
MINLP	Mixed Integer Non-Linear Programming
MNSDE	Modified Non-dominated Sorting Differential Evolution
MOOPs	Multi-Objective Optimization Problems
MPBP	Multi-Product Batch Palnt
MWFB	Method Without forcing the Bound
NAG	Numerical Algorithm Group
NLP	Non-Linear Programming
<i>NPS</i>	Number of solutions in final Pareto Set
NS-1	New Strategy-1
NS-2	New Strategy-2
NSDE	Non-dominated Sorting Differential Evolution
PMP	Pontryagin's Maximum Principle
QN	Quasi-Newton method
R <sub>D</sub>	Rosenbrock function of dimension <i>D</i>
RKFS	Runge – Kutta fixed step size method (fourth order)
RND	Reactor Network Design
SA	Simulated Annealing
SOMA	Self Organizing Migrating Algorithm
SR	Success Rate
SS	Scatter Search
TS	Tabu Search
TS-QN	A hybrid method of Tabu search and Quasi-Newton
Z <sub>D</sub>	Zakharov function of dimension <i>D</i>

---

# CHAPTER 1

---

## INTRODUCTION

The chemical industry has undergone significant changes during the past 25 years due to the increased cost of energy and increasingly stringent environmental regulations. Modification of both plant design procedures and plant operating conditions have been implemented in order to reduce costs and meet the constraints. Most observers both from academia and industry believe that the emphasis in the near future will be on improving efficiency and increasing profitability of existing plants rather than on plant expansion. One of the most important engineering tools that can be employed in such activities is 'optimization'. As computers have become more powerful, the size and complexity of problems which can be simulated and solved by optimization techniques, have correspondingly expanded. Some of the most important examples where optimization involved in chemical industry are: Alkylation Process, Ammonia Synthesis Reactor, Reactor Network Design, Heat Exchanger Network Synthesis & Design, Drying Process, Water Pumping system, Fuel allocation in power plant, Dynamic optimization of chemical processes, Liquid extraction,

membrane design, Parameter estimation, and Optimal control of chemical reactors & Distillation column, etc.

## 1.1. Optimization

Many engineers and researchers face difficulty in understanding the role of optimization in engineering. To many of them, optimization is an esoteric technique used only in mathematics and operations research related activities. A wide variety of problems in the design, construction, operation, and analysis of chemical plants (as well as many other industrial processes) can be resolved by optimization. In fact, it is one of the major quantitative tools in the machinery of decision-making. With the advent of computers, optimization has become a part of computer aided design activities also. The goal of optimization is to find the values of the variables in the process that yield the best value of the performance criterion. What is usually involved is a trade-off between capital and operating costs. Typical problems in chemical engineering design or plant operation have many, and possibly infinite number of, solutions. An important aspect of the optimal design process is the formulation of design problem in a mathematical format that is acceptable to an optimization algorithm. Unfortunately, every optimization problem requires different considerations for formulating objectives, constraints, and variable bounds. Some problems consist of linear terms but many of the problems involve nonlinear terms for constraints and objective function. In a few cases, the terms are not explicit functions of the design variables.

Optimization is concerned with selecting the best among the entire set of possible solutions by efficient quantitative methods. Unfortunately no single method or algorithm of optimization exists that can be applied efficiently to all problems. Some



algorithms perform better on one problem, but may perform poorly on other problems. That is why the literature contains a large number of algorithms, each suitable to a particular type of problem. The method chosen for any particular case will depend primarily on: (1) the character of the objective function and whether it is known explicitly, (2) the nature of the constraints, and (3) the number of independent and dependent variables. Various types of optimization problems are shown in Fig.1.1.

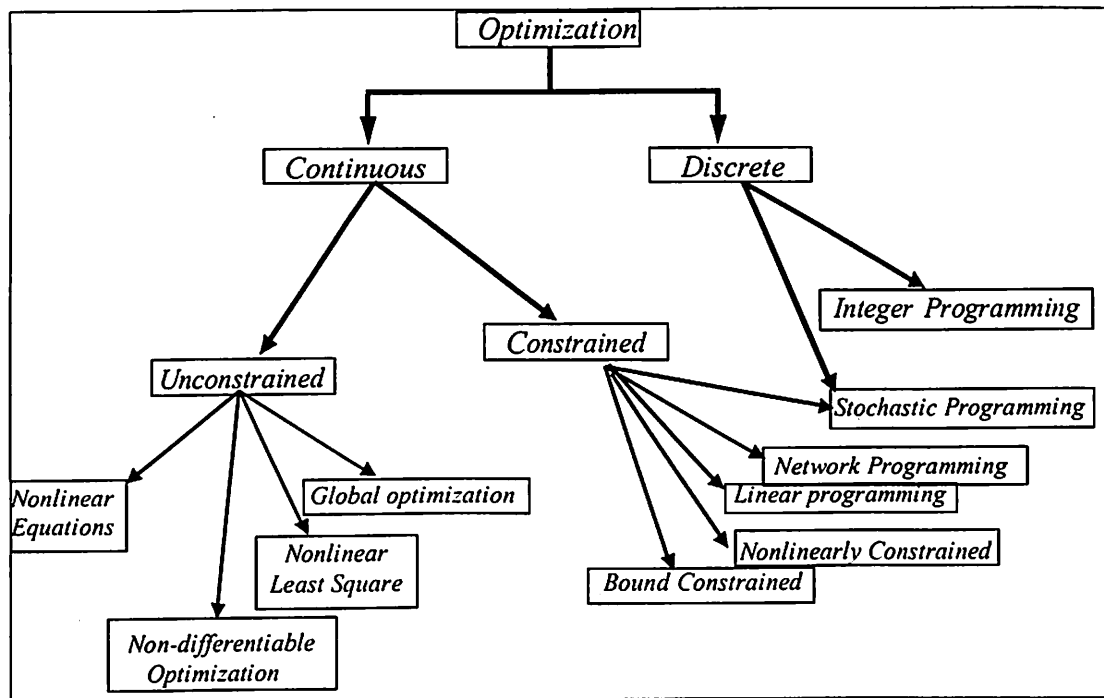


Fig. 1.1. Types of optimization problems

The general objective in optimization is to choose a set of values of the variables subject to various constraints that will produce the desired optimum response for chosen objective function. There are two distinct types of optimization algorithms which are in use today. Firstly, there are algorithms which are deterministic, with specific rules for moving from one solution to the other (also known as traditional methods) which have been successfully applied to some of the engineering design problems. Secondly, there are algorithms, which are stochastic in nature with

probabilistic transition rules that are comparatively new and are gaining popularity due to certain properties, which the deterministic algorithms do not have.

## **1.2. Traditional Methods and their Limitations**

The various traditional methods available for the solution of different types of optimization problems are (Rao, 2003; Babu, 2004): Calculus methods, Calculus of variations, Nonlinear programming, Geometric programming, Quadratic programming, Linear programming, Dynamic Programming, Integer programming, etc. The classical methods of differential calculus can be used to find the unconstrained maxima and minima of a function of several variables. These methods assume that function is differentiable twice with respect to the design variables and the derivatives are continuous. For problems with equality constraints, the Lagrangian multiplier method can be used. If the problem has inequality constraints, the Kuhn-Tucker conditions can be used to identify the optima. But these methods lead to a set of nonlinear simultaneous equations that may be difficult to solve. The methods of nonlinear, linear, geometric, quadratic, or integer programming can be used for the solution of a particular class of problems indicated by the name of the method. Thus, all the traditional methods have their own limitations and applicability. No single method is suitable for all kinds of optimization problems or not even for most of the engineering optimization problems encountered in real world.

Many engineering optimization problems contain multiple optimum solutions, among which one in the case of single objective function and more in the case of multi-objective optimization may be the absolute minimum or maximum solutions. These absolute optimum solutions are known as the global optimal solution and the solutions around and close to the global are called the near global solutions, and the

rest of the optimum solutions are known as local optimal solutions. Ideally, we are interested in the global optimal solutions because they correspond to the absolute optimum objective function value. Most of the traditional optimization algorithms based on gradient methods have the possibility of getting trapped at local optimum depending upon the degree of non-linearity and initial guess. Unfortunately, none of the traditional algorithms are guaranteed to find the global optimal solution (Biegler and Grossmann, 2004), but population based search algorithms are found to have a better global perspective than the traditional methods (Onwubolu and Babu, 2004). Moreover, when an optimal design problem contains multiple global solutions, designers are not only interested in finding just the global optimum solution, but as many near global solutions as possible for various reasons (Edgar and Himmelblau, 1989). Firstly, a design suitable in one situation may not be valid in another situation. Secondly, designers may not be interested in finding the absolute global solution. Instead they are interested in a solution, which corresponds to a marginally inferior objective function value but is more amenable to fabrication. Thus, it is always prudent to know about other equally good solutions for later use. However, if the traditional methods are used to find multiple optimal solutions, they need to be applied a number of times, each time starting from a different initial guess and hoping to achieve a different optimal solution.

In the recent past, non-traditional search and optimization techniques based on natural phenomenon (Evolutionary Computation) such as Genetic Algorithms (GA) (Holland, 1975), Simulated Annealing (SA) (Kirkpatrick et al., 1983), Tabu Search (TS) (Glover, 1989), Memetic Algorithms (MAs) (Norman & Moscato, 1991), Ant colony optimization (ACO) (Dorigo, 1996), Scatter Search (SS) (Glover, 1997), Differential Evolution (DE) (Price & Storn, 1997), and Self Organizing Migrating

Algorithm (SOMA) (Zelinka and Lampinen, 2000), to name a few, have been developed to overcome these problems.

### **1.3. Evolutionary Computation (EC)**

#### **1.3.1. History**

The field began in the late 1950s and early 1960s as the availability of digital computing permitted scientists and engineers to build and experiment with various models of evolutionary processes. Three methodologies have emerged from this early work:

- (i) Evolutionary Programming (EP) (Fogel et al, 1966)
- (ii) Evolution Strategies (ES) (Rechenberg, 1973)
- (iii) Genetic Algorithms (GA) (Holland, 1975).

These served as the basis for much of the work done in 1970s – a period of intense exploration and refinement of these ideas. The result was a variety of robust algorithms with significant potential for addressing difficult scientific and engineering problems. In 1980s each of the subgroups associated with the primary EC paradigms (EP, ESs, and GAs) was involved in planning and holding its own regularly scheduled conferences. However within the field there was growing sense of the need for more interaction and cohesion among the various subgroups. And in 1990s, the field evolved with the name *Evolutionary Computation* (EC).

#### **1.3.2. What is EC?**

**Definition:** *Evolutionary Computation (EC) is the branch of computational intelligence/soft computing that uses computational models of evolutionary processes as key elements in the design and implementation of computer-based problem solving system (Spears et al., 1993).*

There are a variety of evolutionary computational models that have been proposed and studied which we will refer hereafter as EC methods/Evolutionary Algorithms. They share a common conceptual base of simulating the evolution of individual structures via processes of selection and reproduction (discussed in next paragraph). These processes depend on the perceived performance (fitness) of the individual structures as defined by an environment.

More precisely, evolutionary algorithms maintain a population of structures that evolve according to rules of selection and other operators, such as recombination and mutation. Each individual in the population receives a measure of its fitness in the environment. Selection focuses attention on high fitness individuals, thus exploiting the available fitness information. Recombination and mutation perturb those individuals, providing general heuristics for exploration. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

Fig. 1.2 outlines a typical evolutionary algorithm (EA). A population of individual structures is initialized and then evolved from generation to generation by repeated applications of evaluation, selection, recombination, and mutation. The population size  $N$  is generally constant in an evolutionary algorithm, although there is no *a priori* reason (other than convenience) to make this assumption.

An evolutionary algorithm typically initializes its population randomly, although domain specific knowledge can also be used to bias the search. Evaluation measures the fitness of each individual according to its worth in some environment. Evaluation may be as simple as computing a fitness function or as complex as running an elaborate simulation. Selection is often performed in two steps, parent selection and survival. Parent selection decides who become parents and how many children the

parents have. Children are created via recombination that exchanges information between parents, and mutation that further perturbs the children. The children are then evaluated. Finally, the survival step decides who survives in the population.

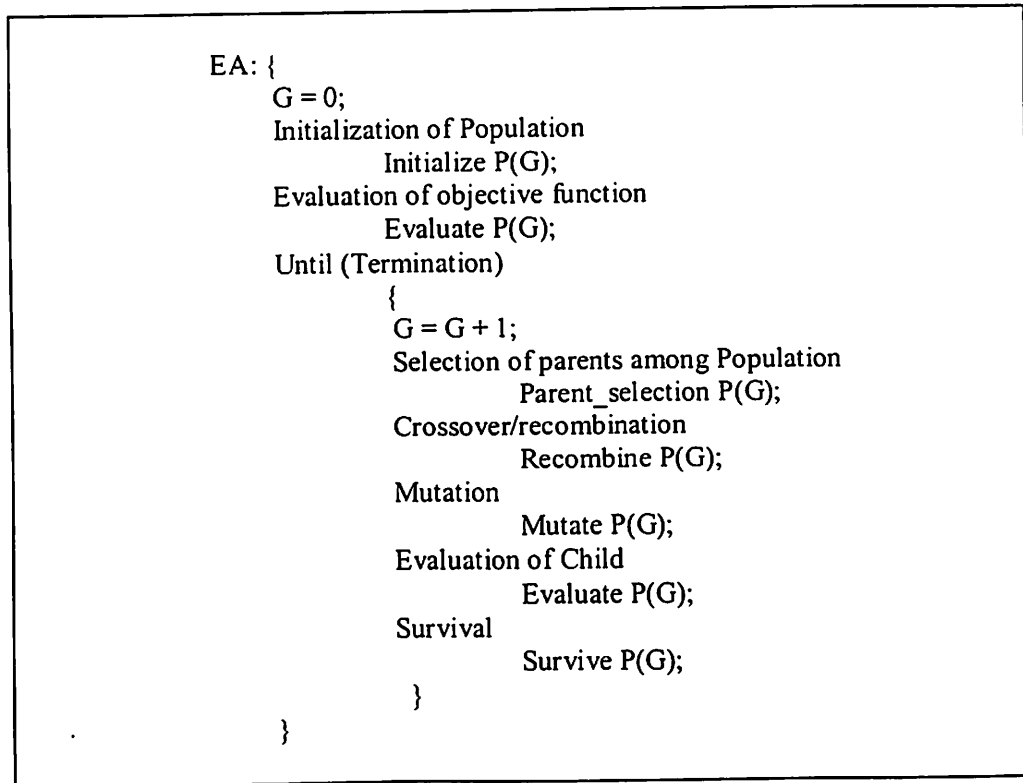


Fig. 1.2. A typical evolutionary algorithm

### 1.3.2.1. Paradigms of EC

The origins of evolutionary algorithms can be traced to as early as 1950's (Fraser, 1957; Box, 1957). Spears et al. (1993) discussed three paradigm of EC that have emerged in the last few decades: "evolutionary programming" (Fogel et al., 1966), "evolution strategies" (Rechenberg, 1973), and "genetic algorithms" (Holland, 1975). They are briefly described below:

Although similar at the highest level, each of these varieties implements an evolutionary algorithm in a different manner. The differences touch upon almost all aspects of evolutionary algorithms, including the choices of representation for the individual structures, types of selection mechanism used, forms of genetic operators,

and measures of performance. We will highlight the important differences (and similarities) in the following sections, by examining the three paradigms of EC.

### 1.3.2.1.1. Evolutionary Programming

Evolutionary Programming (EP), developed by Fogel et al. (1966) traditionally has used representations that are tailored to the problem domain. For example, in real-valued optimization problems, the individuals within the population are real-valued vectors. Similarly, ordered lists are used for traveling salesman problems, and graphs for applications with finite state machines. EP is often used as an optimizer, although it arose from the desire to generate machine intelligence. The outline of the evolutionary programming algorithm is shown in Fig. 1.3.

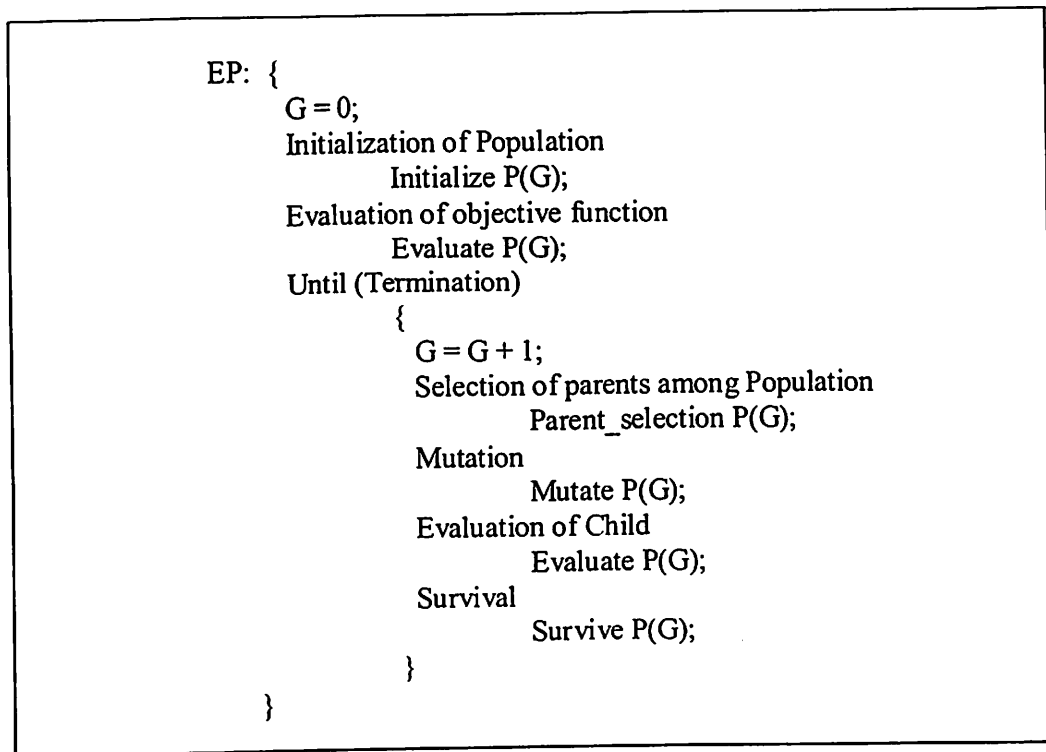


Fig. 1.3. The evolutionary programming algorithm

After initialization, all  $N$  individuals are selected to be parents, and then are mutated, producing  $N$  children. These children are evaluated and  $N$  survivors are chosen from the  $2N$  individuals, using a probabilistic function based on fitness. In other words, individuals with a greater fitness have a higher chance of survival. The

form of mutation is based on the representation used, and is often adaptive. For example, when using a real-valued vector, each variable within an individual may have an adaptive mutation rate that is normally distributed with a zero expectation. Recombination is not generally performed since the forms of mutation used are quite flexible and can produce perturbations similar to recombination, if desired. One of the interesting and open issues is the extent to which an EA is affected by its choice of the operators used to produce variability and novelty in evolving populations.

#### *1.3.2.1.2. Evolution Strategies*

Evolution Strategies (ESs) were independently developed by Rechenberg (1973), with selection, mutation, and a population of size one. Schwefel (1981) introduced recombination and populations with more than one individual, and provided a nice comparison of ESs with many traditional optimization techniques. Due to initial interest in hydrodynamic optimization problems, evolution strategies typically use real-valued vector representations. Fig. 1.4 outlines a typical evolution strategy.

After initialization and evaluation, individuals are selected uniformly and randomly to be the parents. In the standard recombinative ES, pairs of parents produce children via recombination, which are further perturbed via mutation. The number of children created is greater than  $N$ . Survival is deterministic and is implemented in one of two ways. The first allows the  $N$  best children to survive, and replaces the parents with these children. The second allows the  $N$  best children and parents to survive. Like EP, considerable effort has focused on adapting mutation as the algorithm runs by allowing each variable within an individual to have an adaptive mutation rate that is normally distributed with a zero expectation. Unlike EP, however, recombination does play an important role in evolution strategies, especially in adapting mutation.



form of mutation is based on the representation used, and is often adaptive. For example, when using a real-valued vector, each variable within an individual may have an adaptive mutation rate that is normally distributed with a zero expectation. Recombination is not generally performed since the forms of mutation used are quite flexible and can produce perturbations similar to recombination, if desired. One of the interesting and open issues is the extent to which an EA is affected by its choice of the operators used to produce variability and novelty in evolving populations.

#### *1.3.2.1.2. Evolution Strategies*

Evolution Strategies (ESs) were independently developed by Rechenberg (1973), with selection, mutation, and a population of size one. Schwefel (1981) introduced recombination and populations with more than one individual, and provided a nice comparison of ESs with many traditional optimization techniques. Due to initial interest in hydrodynamic optimization problems, evolution strategies typically use real-valued vector representations. Fig. 1.4 outlines a typical evolution strategy.

After initialization and evaluation, individuals are selected uniformly and randomly to be the parents. In the standard recombinative ES, pairs of parents produce children via recombination, which are further perturbed via mutation. The number of children created is greater than  $N$ . Survival is deterministic and is implemented in one of two ways. The first allows the  $N$  best children to survive, and replaces the parents with these children. The second allows the  $N$  best children and parents to survive. Like EP, considerable effort has focused on adapting mutation as the algorithm runs by allowing each variable within an individual to have an adaptive mutation rate that is normally distributed with a zero expectation. Unlike EP, however, recombination does play an important role in evolution strategies, especially in adapting mutation.

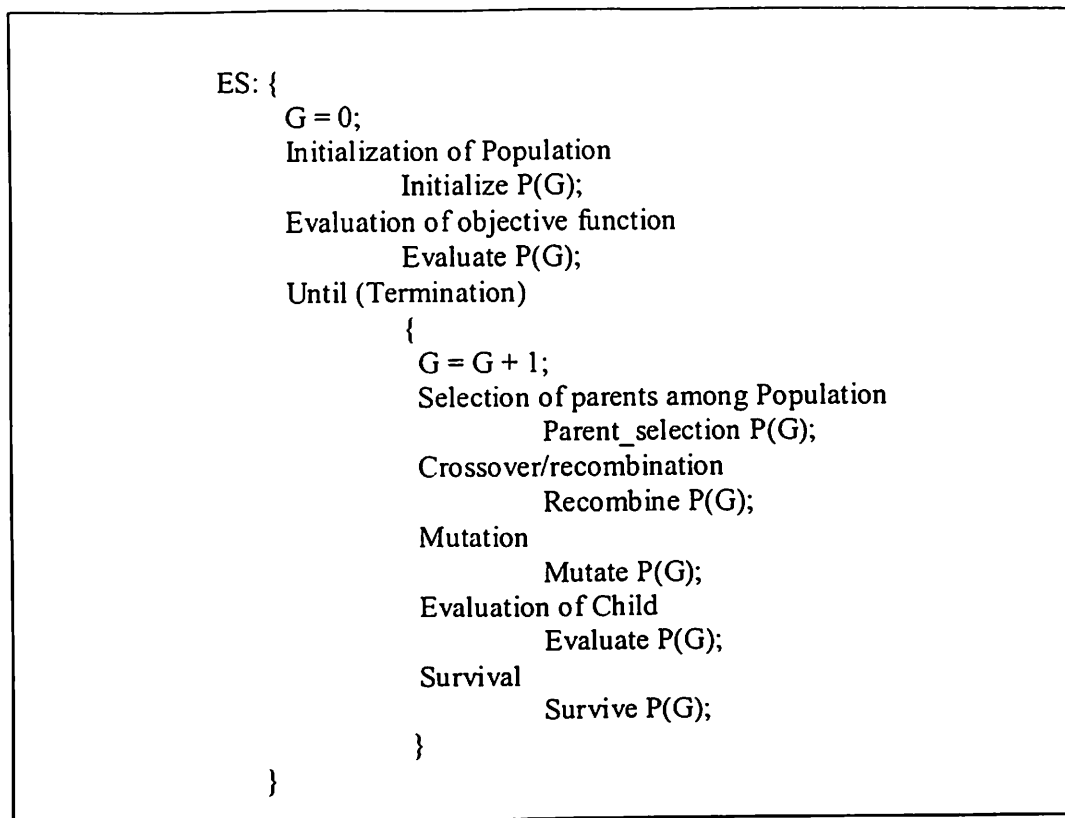


Fig. 1.4. The evolution strategy algorithm

### 1.3.2.1.3. Genetic Algorithms

Genetic Algorithms (GAs), developed by Holland (1975) and subsequently popularized by Goldberg (1989), have traditionally used a more domain independent representation, namely, bit-strings. However, many recent applications of GAs have focused on other representations, such as graphs (neural networks), LISP expressions, ordered lists, and real-valued vectors. Fig. 1.5 outlines a typical genetic algorithm.

After initialization, parents are selected according to a probabilistic function based on relative fitness. In general, a fitness function  $F(x)$  is first derived from the objective function  $f(x)$  and used in successive genetic operations. Certain genetic operators require that the fitness function be nonnegative, although certain operators do not have this requirement. For maximization problems, the fitness function can be considered to be the same as the objective function or  $F(x) = f(x)$ . For minimization problems, the fitness function is an equivalent maximization problem chosen such

that the optimum point remains unchanged. A number of such transformations are possible. The following fitness function is often used:  $F(x) = 1/(1 + f(x))$ . In other words, those individuals with higher relative fitness are more likely to be selected as parents.  $N$  children are created via recombination/crossover from the  $N$  parents. The  $N$  children are mutated and survive, replacing the  $N$  parents in the population.

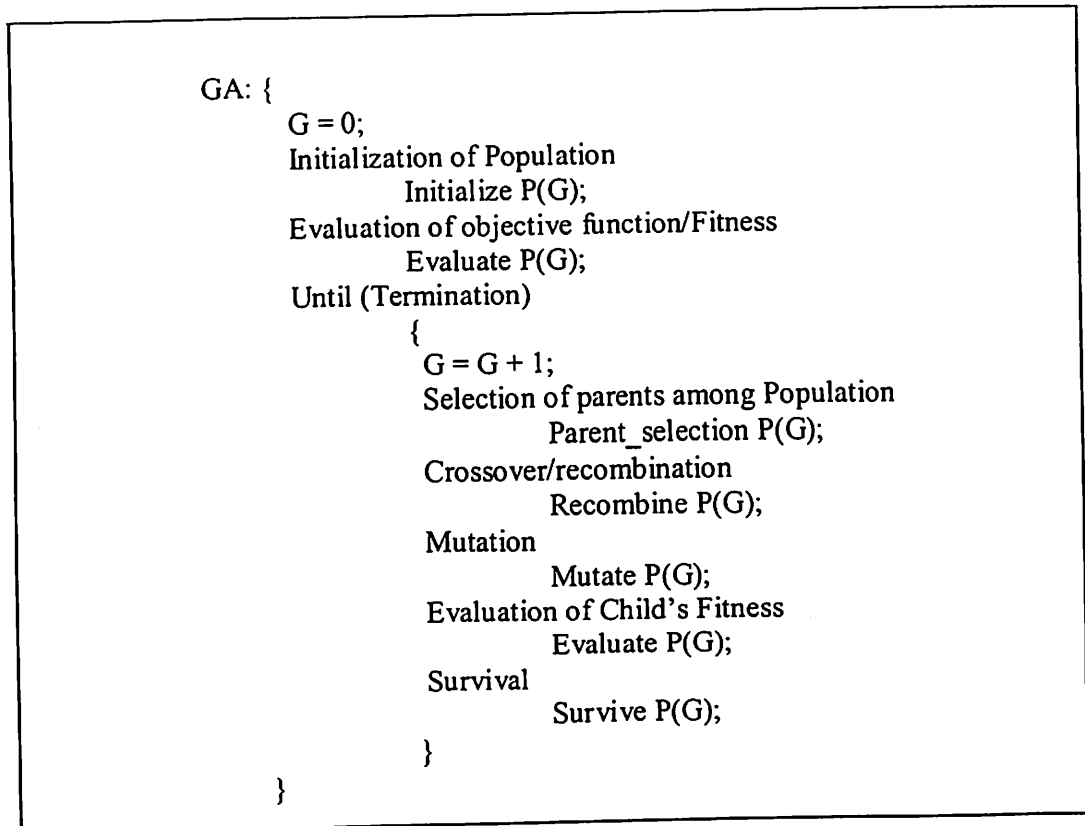


Fig. 1.5. The genetic algorithm

It is interesting to note that the relative emphasis on mutation and crossover is opposite to that in EP. In simple GA, mutation flips bits with some small probability, and is often considered to be a background operator. Recombination, on the other hand, is emphasized as the primary search operator. GAs are often used as optimizers, although some researchers emphasize its general adaptive capabilities (De Jong, 1992).

These three approaches (EP, ESs, and GAs) have served to inspire an increasing amount of research on and development of new forms of evolutionary algorithms for

use in specific problem solving contexts. Some of the upcoming new optimization techniques with their applications in engineering and management are well documented in literature (Onwubolu and Babu, 2004; Corne et al., 1999).

### 1.3.3. Why EC?

As a recognized field, EC is quite young. The term itself was invented recently (in 1991) and it represents an effort to bring together researchers who have been following different approaches to simulating various aspects of evolution. Efforts in evolutionary computation commonly derive from one of four different motivations, namely optimization, robust adaptation, machine intelligence, and understanding of biology.

#### 1.3.3.1. *Optimization*

The classic techniques of gradient descent, deterministic hill climbing, and purely random search have been generally unsatisfactory when applied to non-linear optimization problems. But these are the problems that nature has solved so very well. Evolution provides inspiration for computing the solutions to problems that have previously appeared intractable. This was a key foundation for the efforts in evolution strategies (Rechenberg, 1973; Schwefel, 1995).

#### 1.3.3.2. *Robust adaptation*

The real world is never static, and problems of temporal optimization are some of the most challenging that one would come across. They require the changing behavioral strategies in light of the most recent feedback concerning the success or failure of the current strategy. Holland (1975) developed the framework of genetic algorithms that has the potential to adjust performance based on feedback from the environment.

### **1.3.3.3. *Machine intelligence***

Intelligence may be defined as the capability of a system to adapt its behavior to meet desired goals in a range of environments (Fogel, 1995). Evolution has created creatures of increasing intelligence over time and an alternate approach to generating machine intelligence is to simulate evolution on a class of predictive algorithms. This was the foundation for the evolutionary programming (Fogel et al., 1966).

### **1.3.3.4. *Understanding of biology***

Rather than attempting to use evolution as a tool to solve a particular engineering problem, there is a desire to capture the essence of evolution in a computer simulation to gain new insight into physics of natural evolutionary processes (Ray, 1991). Success raises the possibility of studying alternate biological systems that are merely plausible images of what life might be like in some way. Computer simulations under the rubric of *artificial life* have generated some patterns that appear to correspond with naturally occurring phenomena.

## **1.4. EC vs. Traditional Methods**

The EC and traditional methods are compared by bringing out the differences between them as listed below:

1. Traditional methods starts with a single point while EC methods with a population of points
2. Traditional methods have fixed transition rules while EC methods have probabilistic rules
3. Traditional methods may get trapped at local optima but EC methods are more likely to give global optima

4. Traditional methods are not suitable for multimodal and discrete problems while EC methods are most suited to these kind of problems
5. Traditional methods are to be used again & again to obtain multiple optima causing extra computational effort. EC methods give multiple optima in a single run.

## **1.5. Motivation**

In 1980, optimization of engineering problems beyond linear programming was often viewed as a curious novelty without much benefit. Now optimization applications are essential in all areas of process systems engineering including design, identification, control, estimation, scheduling and planning (Grossmann and Biegler, 2004). The optimization has become a major enabling area in process systems engineering. It has evolved from a methodology of academic interest into a technology that has and continues to make a significant impact in industry. One of the major limitations of the NLP and MINLP methods is that they are not guaranteed to find the global optimum for nonconvex problems (Biegler and Grossmann, 2004). The deterministic methods for nonconvex continuous optimization include, for instance (1) Lipschitzian methods (Hansen et al., 1992); (2) branch and bound methods (Al-Khayyal, 1992; Al-Khayyal and Falk, 1983); (3) cutting plane methods (Tuy, Thieu, and Thai, 1985); (4) outer-approximation (OA) methods (Horst, Thoi, and De Vries, 1992); (5) primal dual methods (Floudas and Visweswaran, 1990); (6) reformulation – linearization methods (Sherali and Alameddine, 1992; Sherali and Tuncbilek, 1992); and (7) interval methods (Hansen, 1980).

Other global optimization approaches for addressing nonconvexities in NLP and MINLP problems assumes special structure in the continuous terms (e.g. bilinear,

linear fractional, concave separable). When combined with global optimization techniques for continuous variables (Falk & Soland, 1969; Quesada and Grossmann, 1995; Ryoo and Sahinidis, 1995; Zamora and Grossmann, 1999), they take the form of spatial branch and bound methods (Soland, 1971). Adjiman et al. (1997, 2000) proposed the SMIN- $\alpha$ BB and GMIN- $\alpha$ BB algorithms for twice differentiable nonconvex MINLPs. All of these methods rely on a spatial branch and bound procedure. The difference lies in how to perform the branching on the discrete and continuous variables. These methods are rigorous and computationally expensive. Also, special reformulation strategies such as the ones that are outlined in Smith and Pantelides (1999) are needed in order to handle algebraic models in terms of basic functional forms (bilinear, fractional, concave separable).

As mentioned earlier, most of the traditional optimization techniques based on gradient methods have the possibility of getting trapped at local optimum depending upon the degree of non-linearity and initial guess. Hence, these traditional optimization techniques do not ensure global optimum and also have limited applications. Now-a-days, non-traditional search and optimization methods (also called Evolutionary Algorithms) that often rely on physical analogies in order to generate trial points that mimic the approach to an equilibrium condition, are becoming popular. Examples include Simulated Annealing (SA) (Kirkpatrick et al., 1983), Genetic Algorithms (GA) (Goldberg, 1989; Davis, 1991) and Differential Evolution (DE) (Price and Storn, 1997), to name a few. These methods are relatively simple and easy to use. Furthermore, these methods are not rigorous and often produce good optimal solutions. Evolutionary Algorithms (EAs) have been widely used in science and engineering for solving complex problems. An important goal of research on evolutionary algorithms is to understand the class of problems for which

EAs are most suited, and, in particular, the class of problems on which they outperform other search algorithms.

Simulated Annealing (SA) is a probabilistic nontraditional optimization technique, which resembles the thermodynamic process of cooling of molten metals to achieve the minimum energy state. Rutenbar (1989) gave a detailed discussion of the working principle of SA and its applications. Since its introduction SA has diffused widely into many diverse applications.

Genetic Algorithms (GAs) are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. They mimic the 'survival of the fittest' principle of nature to make a search process. The key parameters of control in GA are:  $N$ , the population size;  $p_c$ , the crossover probability; and  $p_m$ , the mutation probability (Goldberg, 1989; Deb, 1996). Since their inception, GAs have evolved like the species they try to mimic and have been applied successfully in many diverse fields.

Differential Evolution (DE), a recent optimization technique, is an exceptionally simple evolution strategy, which is significantly faster & robust at numerical optimization and is more likely to find a function's true global optimum (Price and Storn, 1997). Simple GA uses a binary coding for representing problem parameters whereas DE uses real coding of floating point numbers. Among the DE's advantages are its simple structure, ease of use, speed and robustness. It can be used for optimizing functions with real variables and many local optima.

There are many chemical processes which are highly nonlinear and complex with reference to optimal operating conditions with many equality and inequality constraints; such as optimization of a Thermal Cracker, optimal operation of



Alkylation Unit, optimal design of Ammonia Reactor, optimization of Reactor Networks, optimal design of Heat Exchanger Networks, etc.

But application of Differential Evolution (DE) to chemical processes is scarce (Wang and Chiou, 1997; Babu and Sastry, 1999; Lee et al., 1999; Chiou and Wang, 1999; Babu and Munawar, 2000; Babu and Munawar, 2001; Lu and Wang, 2001). Babu and Sastry (1999) used it for the estimation of effective heat transfer parameters in trickle bed reactors using radial temperature profile measurements. They concluded that DE takes less computational time to converge as compared to existing techniques without compromising on the accuracy of the parameter estimates. Earlier DE dealt with a single strategy (Price and Storn, 1997). Later on ten different strategies have been suggested by Price and Storn (2005). A strategy that works out to be the best for a given problem may not work well when applied for a different problem. The strategy to be adopted for each problem is to be determined separately by trial & error.

This study demonstrates the successful application of Differential Evolution (DE), an improved version of GA, to benchmark test functions followed by the selected chemical engineering problems (nonlinear chemical processes, process synthesis and design problems, dynamic optimization problem). Also improvements on original DE are explored. An extension of DE for multi-objective optimization is studied.

## **1.6. Objectives**

With the above background and identified need, the objectives for the present work are set as follows:

- i. To identify and understand some of the highly non-linear chemical processes with respect to optimal operating conditions with equality & inequality constraints.
- ii. To apply Differential Evolution (DE) to all the identified and selected chemical processes of varied nature.
- iii. To get more insight into the fundamental understanding of non-traditional optimization algorithms (like GA and DE), particularly DE and finding out the possibility of improving the performance of existing DE by proposing new strategies or modifications in original DE.
- iv. To compare the performance of DE with other traditional/evolutionary optimization methods for the selected problems.
- v. To extend DE for solving multi-objective optimization problems.

### **1.7. Scope of Present Study**

Keeping the objectives mentioned above in view, the present research work is formulated to study the following aspects:

1. Problem formulation for selected non-linear chemical processes.
2. Study of DE in detail to understand the intricacies of the algorithm. And the question that what makes DE so powerful is quite interesting and obvious to explore.
3. Understanding the ten DE strategies.
4. Application of DE to the identified & selected non-linear chemical processes (which represents various types of optimization problems such as Linear programming problems, Non-linear programming problems, Process synthesis

and design (mixed integer non-linear programming) problems, Dynamic optimization problems etc.).

5. Seeking improvements on DE by finding better mutation and crossover schemes or by hybridizing it with traditional methods.
6. Comparison of the performance of DE for a given problem with that of other Traditional/Evolutionary optimization methods.
7. Extension of DE for solving multi-objective optimization problems.

## 1.8. Structure of the Thesis

The subsequent chapters of this thesis are arranged as follows:

**Chapter-2** explains briefly about GA. Further the working principle of DE in detail by applying it to a simple function. Also, the performance of GA is compared with that of DE using a benchmark test function.

In **Chapter-3**, a modification in the original DE is proposed in order to enhance its performance. Modified DE and original DE algorithms have been applied to several benchmark test functions. Also the effect of DE and MDE parameters on their performance is studied using two benchmark test functions. Selected nonlinear chemical processes are explained. And MDE and DE are used to solve these selected problems. The performance of DE is compared with that of the proposed MDE and other methods like GA, hill climbing method, direct search method, controlled random search method, geometric programming, and  $\alpha$ BB algorithm (by which the particular problem has been solved in literature)

**Chapter-4** deals with the application of DE and MDE to some process synthesis and design problems (which are generally Mixed Integer Non-Linear Programming

problems) encountered in chemical engineering. Performance of DE is compared with that of MDE, GA, ES, and MINLP Simplex Simulated Annealing (M-SIMPSA).

In **Chapter-5**, a new hybrid DE (HDE) algorithm proposed in the present study is discussed. Further, the performance of HDE algorithm is evaluated by its application to various chemical engineering problems and comparing its performance with that of DE.

Two new strategies of DE proposed in the present study are discussed in detail in **Chapter-6**. The control parameters of new strategies are tuned using two test functions. Also, the two strategies have been applied to selected difficult nonlinear chemical engineering problems and their performance is compared with that of DE and MDE.

The extension of DE and MDE to solve Multi-Objective Optimization Problems (MOOPs) is discussed in **Chapter-7**. Two novel techniques to solve MOOPs, *Non-dominated Sorting Differential Evolution* (NSDE) and *Modified Non-dominated Sorting Differential Evolution* (MNSDE), proposed in this study are described in detail with the help of two benchmark test problems. Further the effect of control parameters on the performance of NSDE and MNSDE is investigated.

Finally, in the **Chapter-8**, a brief summary of the present work is presented followed by conclusions, major contributions, and future scope for research. Codes of the selected problems from the present study are given in appendices.

## EVOLUTIONARY COMPUTATION METHODS

During the last two decades there has been a growing interest in optimization algorithms, which are based on the principle of evolution (survival of the fittest). A common term, accepted recently, refers to such techniques as Evolutionary Algorithms (EA) or Evolutionary Computation methods (EC methods). The best-known algorithms in this class include Genetic Algorithms, Evolutionary Programming, Evolution Strategies, and Genetic Programming. There are many hybrid systems, which incorporate various features of the above paradigms and consequently are hard to classify, which can be referred just as EC methods (Dasgupta and Michalewicz, 1997). Out of the three paradigms of EC, GA (proposed by Holland, 1975) has been found to be the most popular algorithm. Differential Evolution (Price and Storn, 1997), due to its robustness, speed and simplicity is chosen for application in the present study. Also, a comparison made with GA using a test function further substantiate the fact that DE is better than simple GA. Further, other algorithms, proposed in the present study, i.e., Modified Differential Evolution

(MDE), Hybrid Differential Evolution (HDE), and two new strategies of differential evolution are discussed in Chapter-3, 5, and 6 respectively. In this chapter first genetic algorithm is explained briefly and then differential evolution is described in detail. Further, a non-linear function is used to evaluate the performance of the two evolutionary optimization techniques.

## 2.1. Genetic Algorithms (GAs)

GAs are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. They mimic the ‘survival of the fittest’ principle of nature to make search process. The key parameters of control are:  $N$  - the population size,  $p_c$  - the crossover probability and  $p_m$  - the mutation probability (Goldberg, 1989). The operation of GAs begins with a population of random strings representing design or decision variables. Thereafter, each string is evaluated. The population is then operated by three main operators (1) reproduction, (2) crossover, and (3) mutation in order to create a new population of points. The new population is further evaluated and termination criterion is checked. If the termination criterion is not met, the population is subjected to the above three operators iteratively and evaluated. This procedure is continued until termination criterion is fulfilled.

The three operators mentioned above are simple and straightforward. The reproduction operator selects good strings and the crossover operator recombines good substrings from good strings together to hopefully create a better string. Even though none of these claims are guaranteed and/or tested while creating a string, it is expected that if bad strings are created they will be eliminated by the reproduction operation in the next generation and if the good strings are produced, they will be increasingly emphasized to take part in the further generations. In finding a global

optimum usually both the convergence and divergence are equally important in any population-based optimization algorithm. Divergence from the point of view of not to miss any point in the search space via extensive global search and convergence from the view point of getting closer to the global solution via extreme local search. In general, crossover and mutation takes care of the above two important aspects of divergence and convergence respectively.

### 2.1.1. Stepwise Procedure of GA

The various steps involved in GA are as follows:

- Step-1.** Choose the problem parameters, a selection operator, a crossover operator, and a mutation operator. Choose the control parameters population size,  $N$ , crossover probability,  $p_c$ , and mutation probability,  $p_m$ . Initialize a random population of strings of size  $\ell$ . Choose a maximum allowable generation number  $gen\_max$ . Set  $gen = 0$ .
- Step-2.** Evaluate each string in the population.
- Step-3.** if  $gen > gen\_max$  or other termination criterion is satisfied, **Terminate and print results.**
- Step-4.** Perform reproduction on the population.
- Step-5.** Perform crossover on the randomly chosen pairs of strings.
- Step-6.** Perform mutation on every string.
- Step-7.** Evaluate strings in the new population. Set  $gen = gen + 1$ , and go to Step-3.

The algorithm is straightforward and is repeated application of three operators to a population of points.

### 2.1.2. Applications of GA

Since their inception, GAs have been applied in many fields. The various applications of GAs are: process design and optimization (Androulakis and

optimum usually both the convergence and divergence are equally important in any population-based optimization algorithm. Divergence from the point of view of not to miss any point in the search space via extensive global search and convergence from the view point of getting closer to the global solution via extreme local search. In general, crossover and mutation takes care of the above two important aspects of divergence and convergence respectively.

### 2.1.1. Stepwise Procedure of GA

The various steps involved in GA are as follows:

- Step-1.** Choose the problem parameters, a selection operator, a crossover operator, and a mutation operator. Choose the control parameters population size,  $N$ , crossover probability,  $p_c$ , and mutation probability,  $p_m$ . Initialize a random population of strings of size  $\ell$ . Choose a maximum allowable generation number  $gen\_max$ . Set  $gen = 0$ .
- Step-2.** Evaluate each string in the population.
- Step-3.** if  $gen > gen\_max$  or other termination criterion is satisfied, **Terminate and print results.**
- Step-4.** Perform reproduction on the population.
- Step-5.** Perform crossover on the randomly chosen pairs of strings.
- Step-6.** Perform mutation on every string.
- Step-7.** Evaluate strings in the new population. Set  $gen = gen + 1$ , and go to Step-3.

The algorithm is straightforward and is repeated application of three operators to a population of points.

### 2.1.2. Applications of GA

Since their inception, GAs have been applied in many fields. The various applications of GAs are: process design and optimization (Androulakis and



optimum usually both the convergence and divergence are equally important in any population-based optimization algorithm. Divergence from the point of view of not to miss any point in the search space via extensive global search and convergence from the view point of getting closer to the global solution via extreme local search. In general, crossover and mutation takes care of the above two important aspects of divergence and convergence respectively.

### 2.1.1. Stepwise Procedure of GA

The various steps involved in GA are as follows:

- Step-1.** Choose the problem parameters, a selection operator, a crossover operator, and a mutation operator. Choose the control parameters population size,  $N$ , crossover probability,  $p_c$ , and mutation probability,  $p_m$ . Initialize a random population of strings of size  $\ell$ . Choose a maximum allowable generation number  $gen\_max$ . Set  $gen = 0$ .
- Step-2.** Evaluate each string in the population.
- Step-3.** if  $gen > gen\_max$  or other termination criterion is satisfied, **Terminate and print results.**
- Step-4.** Perform reproduction on the population.
- Step-5.** Perform crossover on the randomly chosen pairs of strings.
- Step-6.** Perform mutation on every string.
- Step-7.** Evaluate strings in the new population. Set  $gen = gen + 1$ , and go to Step-3.

The algorithm is straightforward and is repeated application of three operators to a population of points.

### 2.1.2. Applications of GA

Since their inception, GAs have been applied in many fields. The various applications of GAs are: process design and optimization (Androulakis and

Venkatasubramanian, 1991), computer aided molecular design (Venkatasubramanian et al., 1994), synthesis & optimization of non-ideal distillation systems (Fraga and Matias, 1996), optimal design of ammonia synthesis reactor (Upreti and Deb, 1996), molecular scale catalyst design (McLeod et al., 1997), estimation of heat transfer parameters in trickle bed reactors (Babu and Vivek, 1999), automated design of heat exchangers using artificial intelligence based optimization (Babu and Mohiddin, 1999), optimal design of heat exchangers (Manish et al., 1999) etc. to name a few.

## 2.2. Differential Evolution (DE)

Price and Storn (1997) have given the working principle of Differential Evolution (DE), which is an improved version of GA, along with its application to polynomial fitting problems. They have also suggested some simple rules for choosing the key parameters such as *NP*- the population size, *CR*- crossover constant and *F*- the weight applied to random differential (scaling factor) of DE for any given application. In their web site, Price and Storn (2005) have given the various recent applications of DE along with the program codes in various computer languages (C, C++, Matlab, Fortran90, Scilab, and Python). They also suggested ten different working strategies of DE and some guidelines in applying these strategies for any given problem. Different strategies can be adopted in DE algorithm depending upon the type of problem for 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 proposed by Price and Storn (2005):

- I. DE/best/1/exp
- II. DE/rand/1/exp
- III. DE/rand-to-best/1/exp

- IV. DE/best/2/exp
- V. DE/rand/2/exp
- VI. DE/best/1/bin
- VII. DE/rand/1/bin
- VIII. DE/rand-to-best/1/bin
- IX. DE/best/2/bin
- X. 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). Thus, the working algorithm outlined by Price and Storn (1997) is the seventh strategy of DE, i.e., DE/rand/1/bin. 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  $D$  (the dimension, i.e., number of variables to be optimized) variables in one loop until it is within the  $CR$  bound. For discrete optimization problems (such as Design of shell & tube heat exchanger), 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 (Babu and Munawar, 2001). 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 crossovers yield similar results. The strategy to be adopted for each problem is to be determined separately by trial and error. A strategy that works out to be the best for a given problem may not work well when applied for a different problem. The algorithm of DE is given below:

- Choose a strategy
- Initialize the value of  $D$ ,  $NP$ ,  $CR$ ,  $F$  &  $gen\_max$ .
- Initialize all the vector population randomly in the given upper & lower bound.

For  $i = 1$  to  $NP$

{For  $j = 1$  to  $D$

$X_{ij} = \text{random Number}$ }

- Evaluate the cost of each vector.

*Repeat*

Perform mutation, crossover, selection & evaluation of the objective function for a specified number of generations.

- (a). For each vector  $X_i$  (target vector), select three distinct vectors  $X_a$ ,  $X_b$  &  $X_c$  (select five, if two vector differences are to be used) randomly from the current population (primary array) other than vector  $X_i$ .
- (b). Perform crossover for each target vector with its noisy vector to create a trial vector.
- (c). Perform selection for each target vector,  $X_i$  by comparing its cost with that of the trial vector. Vector with lower cost is selected for next generation.

*Till termination criteria do not meet.*

- Print results.

Choosing the values of  $NP$ ,  $F$ , and  $CR$  depends on the specific problem applied, and is often difficult. But some general guidelines are available. Generally,  $NP$  should be about 5 to 10 times the number of parameters in a vector. As for  $F$ , it lies in the range of 0.4 to 1.0. Initially  $F = 0.5$  can be tried and then  $F$  and/or  $NP$  is increased if the population converges prematurely. A good first choice for  $CR$  is 0.1, but in general  $CR$  should be as large as possible (Price and Storn, 1997). The best combination of these key parameters of DE for each of the strategies mentioned earlier is again different. Price and Storn (2005) have mentioned some simple rules for choosing the best strategy as well as the corresponding key parameters. Among

DE's advantages are its simple structure, ease of use, speed and robustness. The schematic diagram of working method of DE is shown in Fig. 2.1.

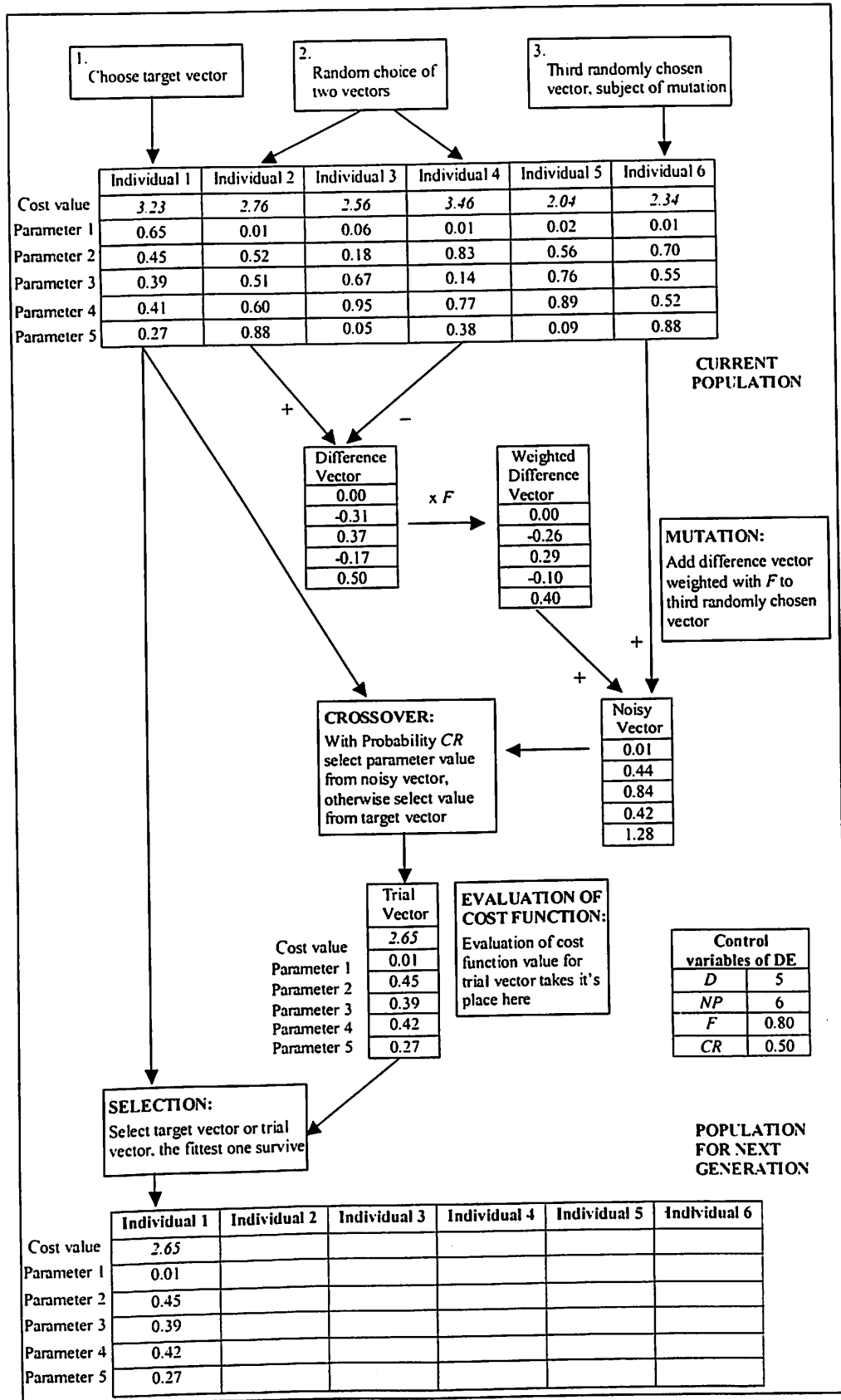


Fig. 2.1. Schematic of working of DE

### 2.2.1. Working of DE

The working of each step of the above algorithm is described in detail with the help of an example.

Consider the following function for minimization:

$$f(x) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2; \quad 0 \leq x \leq 6;$$

**Step-1. Initialize population** between given upper and lower bound for all parameters.

**Step-2. Evaluate:** Calculate objective function for initial population (Column-4 in Table-2.1).

**Table-2.1. Initial Population**

S. No	$x_1$	$x_2$	cost
0	3.677337	0.222648	32.187374
1	2.338032	3.289922	36.188379
2	0.528440	5.870270	35.880912
3	1.187137	4.523740	30.686806
4	3.133475	4.677932	69.292287
5	3.216689	2.511420	43.863578
6	0.447552	1.098901	1.940472
7	4.642198	1.944511	67.632426
8	2.412028	0.662900	17.022193
9	5.517039	5.075050	143.072049
10	4.422031	4.354949	96.656809
11	3.360381	3.322859	55.995388
12	4.689647	2.206954	72.038578
13	4.020225	3.727019	76.475217
14	4.645876	2.039149	68.880441
15	4.023861	3.511431	73.484507
16	5.000684	5.525994	135.292475
17	3.093430	4.681739	68.434256
18	4.498183	2.810695	75.340828
19	5.460903	3.884746	118.738774

In Table-2.1,  $x_1$  and  $x_2$  are parameters and cost is objective function  $f(x)$ .

**Step-3. Mutation & Cross over:**

In Table-2.2,  $i$  is population counter  $i = (0, 1, 2, \dots, 19)$ . See column-1.

- a. Randomly choose three population points  $a, b, c$  such that  $i \neq a \neq b \neq c$  (see column-2, 3 and 4 in Table-2.2).
- b. Select randomly a parameter  $j$  for mutation, Column-5 in Table-2.2, ( $j=0,1$ ).
- c. Generate a random number  $[0, 1]$ . (Column-6, 7 in Table-2.2)

If this is less than  $CR$  (0.5), then mutate the parameter as shown in the equation

below:

$$\text{trial}[j] = X1[c][j] + F*(X1[a][j] - X1[b][j]); \text{ (Where } F = 0.8\text{)}$$

If the random number generated is not less than  $CR$ , then take the parameter as shown in the equation below:

$$\text{trial}[j] = X1[i][j]; \quad \text{(Column-8, 9 in Table-2.2)}$$

Check for bounds; if bounds are violated, then randomly generate the parameter as shown below:

$$\text{trial}[j] = \text{lower\_limit} + \text{rand-no.}[0,1]*(\text{upper\_limit} - \text{lower\_limit});$$

(as shown in column-10, 11 in Table-2.2) ( $\text{lower\_limit} = 0.0$  and  $\text{upper\_limit} = 6.0$ )

Repeat step-3 till all parameters are mutated. Here we have two parameters.

**Step-4. Evaluation:** Calculate the objective function value for vector obtained after mutation and crossover (see column 12 in Table-2.2).

**Step-5. Selection:** Select the least cost vector for next generation (if problem is of minimization. (See Column-2, 3 and 4 in Table-2.3).

**Step-6.** Repeat steps-3 to 5 for a specified number of generations or till some termination criterion is met.

The optimum is at:  $x_1 = 0.0$ ; and  $x_2 = 0.5$ ; with  $f(x) = -0.25$ ;

For the above problem, the termination criterion used is  $(\text{Costmax} - \text{Costmin}) \leq 10^{-7}$ .  $\text{Costmax}$  is maximum value of objective function and  $\text{Costmin}$  is minimum value

of objective function in a generation of population. The number of function evaluations is found to be 1960 to obtain the optimum as stated above.

**Table-2.2. Mutation and Crossover**

i	a	b	c	j	Rand-no. [0, 1]	Rand-no. [0, 1]	$x_1$	$x_2$	$x_1$	$x_2$	Cost
0	8	14	7	1	0.46021	0.699338	2.855120	0.843512			23.843199
1	14	15	3	1	0.894391	0.5914	1.684749	3.289922			25.980564
2	4	6	16	0	0.31743	0.260761	7.149422	8.389219	1.935486	1.109460	13.843822
3	2	7	6	1	0.381619	0.315992	-2.84345	4.23951	4.197434		88.758375
4	15	11	13	1	0.058299	0.882461	4.551009	3.877877			92.430927
5	19	15	0	1	0.659134	0.978825	4.826971	2.511420			79.467166
6	15	10	14	1	0.115894	0.613161	4.327340	1.364335			54.084030
7	1	19	3	0	0.086053	0.824881	-1.31116	4.047881	2.563926		48.805758
8	3	18	13	1	0.095357	0.457195	1.371389	5.097455			40.000583
9	19	16	17	0	0.006763	0.796946	3.461605	3.368741			58.729204
10	16	11	7	0	0.214108	0.073992	5.954440	3.707019			131.046566
11	15	2	9	1	0.845929	0.354696	8.313375	3.322859	3.239810		53.481945
12	2	6	10	0	0.781496	0.255899	4.689647	8.172045		2.577234	76.912755
13	9	19	17	1	0.062814	0.146946	3.138339	5.633982			84.307141
14	8	3	5	1	0.564136	0.54419	4.196601	2.039149			58.653499
15	17	11	18	1	0.24459	0.716581	4.284621	3.897799			85.696813
16	11	14	4	1	0.005858	0.551914	2.105079	5.704900			61.827322
17	8	14	7	1	0.604323	0.201341	2.855120	4.681739			63.129346
18	9	7	13	1	0.889923	0.197326	4.720098	2.810695			80.901567
19	0	9	6	1	0.127835	0.064069	-1.02421	-2.78302	3.081357	3.496450	52.347228

**Table-2.3. Next Generation Population**

S. No.	$x_1$	$x_2$	Cost
0	2.855120	0.843512	23.843199
1	1.684749	3.289922	25.980564
2	1.935486	1.109460	13.843822
3	1.187137	4.523740	30.686806
4	3.133475	4.677932	69.292287
5	3.216689	2.511420	43.863578
6	0.447552	1.098901	1.940472
7	2.563926	4.047881	48.805758
8	2.412028	0.662900	17.022193
9	3.461605	3.368741	58.729204
10	4.422031	4.354949	96.656809
11	3.239810	3.322859	53.481945
12	4.689647	2.206954	72.038578
13	4.020225	3.727019	76.475217
14	4.196601	2.039149	58.653499
15	4.023861	3.511431	73.484507
16	2.105079	5.704900	61.827322
17	2.855120	4.681739	63.129346
18	4.498183	2.810695	75.340828
19	3.081357	3.496450	52.347228



### 2.2.2. Applications of DE

Differential Evolution (DE) is an improved version of simple GA. It is exceptionally simple, significantly faster & robust at numerical optimization and is more likely to find a function's true global optimum (Babu and Gautam, 2001; Babu and Munawar, 2000, 2001; Babu and Singh, 2000; Babu and Sastry, 1999; Rudolph, 1996; Zaharie, 2002). DE has been successfully applied in various fields. The various applications of DE include: digital filter design (Storn, 1995), fuzzy decision making problems of fuel ethanol production (Wang et al., 1998), Design of fuzzy logic controller (Sastry et al., 1998), batch fermentation process (Chiou and Wang, 1999; Wang and Cheng, 1999), multi sensor fusion (Joshi & Sanderson, 1999), dynamic optimization of continuous polymer reactor (Lee et al., 1999), estimation of heat transfer parameters in trickle bed reactor (Babu and Sastry, 1999), optimal design of heat exchangers (Babu and Munawar, 2000; 2001), synthesis & optimization of heat integrated distillation system (Babu and Singh, 2000), optimization of an alkylation reaction (Babu and Chaturvedi, 2000), scenario- integrated optimization of dynamic systems (Babu and Gautam, 2001), determining the number of components in mixtures of linear models (Dollena et al., 2001), Identification of hysteretic systems using the differential evolution algorithm (Kyprianou et al., 2001), optimization of Low Pressure Chemical Vapour Deposition Reactors Using Hybrid Differential Evolution (Lu and Wang, 2001), hybrid differential evolution for problems of Kinetic Parameter Estimation and Dynamic Optimization of an Ethanol Fermentation Process (Wang et al., 2001), Tight-Binding Calculations of Si-H Clusters (Chakraborti et al., 2001), Solving Problems Subject to Multiple Nonlinear Constraints (Lampinen, 2001), Co-Evolutionary Hybrid Differential Evolution for Mixed-Integer Optimization Problems (Lin et al., 2001), Optimal Control (Lopez et al., 2001),

Training of Artificial Neural Networks (Ilonen, 2003), Optimization of Pyrolysis of Biomass (Babu and Chaurasia, 2003), etc. Recently, Onwubolu & Babu (2004) compiled new techniques and their applications to various disciplines of engineering and management.

### 2.3. Comparison of GA and DE

A comparison of simple GA and DE is presented in Table-2.4. Comparison is made on the basis of type representation used for decision variables, crossover and mutation operations and the sequence/order of operators used.

**Table-2.4. Comparison of GA and DE**

S. No.	Simple GA	DE
1. Coding	Binary Coding	Real coding
2. Key parameters	$N, p_c$ , and $p_m$	$NP, CR$ , and $F$
3. Mutation	By changing 1 to 0 and vice-versa.	Addition as mutation operator
4. Crossover	Between two randomly chosen strings	Between target & random noisy vector to give trial-vector
5. Selection	Good strings in a Probabilistic way to form mating pool.	Cost of trial & target vector is Compared
6. Order of operation	Selection ↓ Crossover ↓ Mutation	Mutation ↓ Crossover ↓ Selection

### 2.4. Application of DE to a Test Function

In this section, a non-linear function with three local optima and single global optimum is solved using DE. Differential evolution (DE) is used to find the optimal solution of a non-linear function (Himmelblau function), given by:

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2. \quad 0 \leq x \leq 6.$$

The results thus obtained are compared with that of GA (Deb, 1996). Earlier, GA was applied by Deb (1996) to the same function and following results were obtained.

Table-2.5 shows the number of times (out of 100 experiments) that GAs are able to converge to a solution within 1% of global minimum and the average number of function evaluations required to achieve that solution.

**Table-2.5. Results of GA**

$p_c$	$p_m$	$N$	$gen\_max$	$NRC_{GA}$	$NFE_{GA}$
0.8	0.05	20	625	77	426
0.8	0.05	30	416	79	484
0.8	0.05	40	312	76	650

In the present study, DE (with all ten strategies) is applied to find the optimum solution of Himmelblau function and the results are shown in Table-2.6 (key parameters used are  $CR = 0.9$ ,  $F = 0.51$ ,  $NP = 20$ , seed = 5, accuracy = 0.0001%).

**Table-2.6. Results of DE with all ten strategies**

S. No.	Strategy	$G_{min}$	$NFE$
1	DE/rand/1/bin	42	860
2	DE/best/1/bin	22	460
3	DE/best/2/bin	34	700
4	DE/rand/2/bin	54	1100
5	DE/rand-to-best/1/bin	31	640
6	DE/rand/1/exp	40	820
7	DE/best/1/exp	18	380
8	DE/best/2/exp	34	700
9	DE/rand/2/exp	49	1000
10.	DE/rand-to-best/1/exp	29	600

Results of Table-2.6 indicate that the strategy no-2 & 7 are better than other strategies because of less number of function evaluations. Also the accuracy has been increased from 1% to 0.0001% still the numbers of function evaluations are less in case of strategy no. 7 than GA with  $N = 20$ . Table-2.7 & Table-2.8 show the number of times (out of 100 experiments) that strategy no.2 & 7 have been able to converge to a solution within 1% of global minimum and the average number of function evaluations required to achieve that solution (with key parameters) respectively.

Table-2.5 shows the number of times (out of 100 experiments) that GAs are able to converge to a solution within 1% of global minimum and the average number of function evaluations required to achieve that solution.

**Table-2.5. Results of GA**

$p_c$	$p_m$	$N$	$gen\_max$	$NRC_{GA}$	$NFE_{GA}$
0.8	0.05	20	625	77	426
0.8	0.05	30	416	79	484
0.8	0.05	40	312	76	650

In the present study, DE (with all ten strategies) is applied to find the optimum solution of Himmelblau function and the results are shown in Table-2.6 (key parameters used are  $CR = 0.9$ ,  $F = 0.51$ ,  $NP = 20$ , seed = 5, accuracy = 0.0001%).

**Table-2.6. Results of DE with all ten strategies**

S. No.	Strategy	$G_{min}$	$NFE$
1	DE/rand/1/bin	42	860
2	DE/best/1/bin	22	460
3	DE/best/2/bin	34	700
4	DE/rand/2/bin	54	1100
5	DE/rand-to-best/1/bin	31	640
6	DE/rand/1/exp	40	820
7	DE/best/1/exp	18	380
8	DE/best/2/exp	34	700
9	DE/rand/2/exp	49	1000
10.	DE/rand-to-best/1/exp	29	600

Results of Table-2.6 indicate that the strategy no-2 & 7 are better than other strategies because of less number of function evaluations. Also the accuracy has been increased from 1% to 0.0001% still the numbers of function evaluations are less in case of strategy no. 7 than GA with  $N = 20$ . Table-2.7 & Table-2.8 show the number of times (out of 100 experiments) that strategy no.2 & 7 have been able to converge to a solution within 1% of global minimum and the average number of function evaluations required to achieve that solution (with key parameters) respectively.

**Table-2.7. Results of Strategy No.2 (DE/best/1/bin)**

<i>CR</i>	<i>F</i>	<i>NP</i>	<i>gen_max</i>	<i>NRC<sub>DE</sub></i>	<i>NFE<sub>DE</sub></i>	<i>NFE<sub>GA</sub></i>
0.9	0.51	20	625	100	257	426
0.9	0.51	30	416	100	400	484
0.9	0.51	40	312	100	515	650

**Table-2.8. Results of Strategy No.7 (DE/best/1/exp)**

<i>CR</i>	<i>F</i>	<i>NP</i>	<i>gen_max</i>	<i>NRC<sub>DE</sub></i>	<i>NFE<sub>DE</sub></i>	<i>NFE<sub>GA</sub></i>
0.9	0.51	20	625	100	243	426
0.9	0.51	30	416	99	350	484
0.9	0.51	40	312	98	496	650

From Tables-2.7 & Table-2.8, it is evident that number of function evaluations is less in strategy no. 2 & 7 than that of GA. The results of the Table-2.7 show that  $NFE_{DE}$  is less by 39.67%, 17.35%, and 20.77% than  $NFE_{GA}$  for  $NP = 20, 30,$  and  $40$  respectively. Also, the number of runs converged are 100 (out of 100 experiments), i.e., 100% with DE/best/1/bin. But in the case of GA, it is between 76-79 (out of 100 experiments) for different value of population size. And, from Table-2.8, it is clear that  $NFE_{DE}$  is less by 42.96%, 27.69%, 23.69% than  $NFE_{GA}$  for  $NP = 20, 30,$  and  $40$  respectively. As the results of Table-2.7 & Table-2.8 show that strategy no. 7 (DE/best/1/exp) is best because of the least number of function evaluations. The number of experiments converged are 100, 99, and 98 for  $NP = 20, 30,$  and  $40$  respectively.

Fig. 2.2 shows the  $NFE_{DE}/NFE_{GA}$  vs.  $N/NP$  for GA and DE (strategy no. 2 (DE2) & strategy no.7 (DE7)). It is clear that for the three population sizes considered, GA took more number of function evaluations than that of two DE strategies even though the accuracy of GA is low.

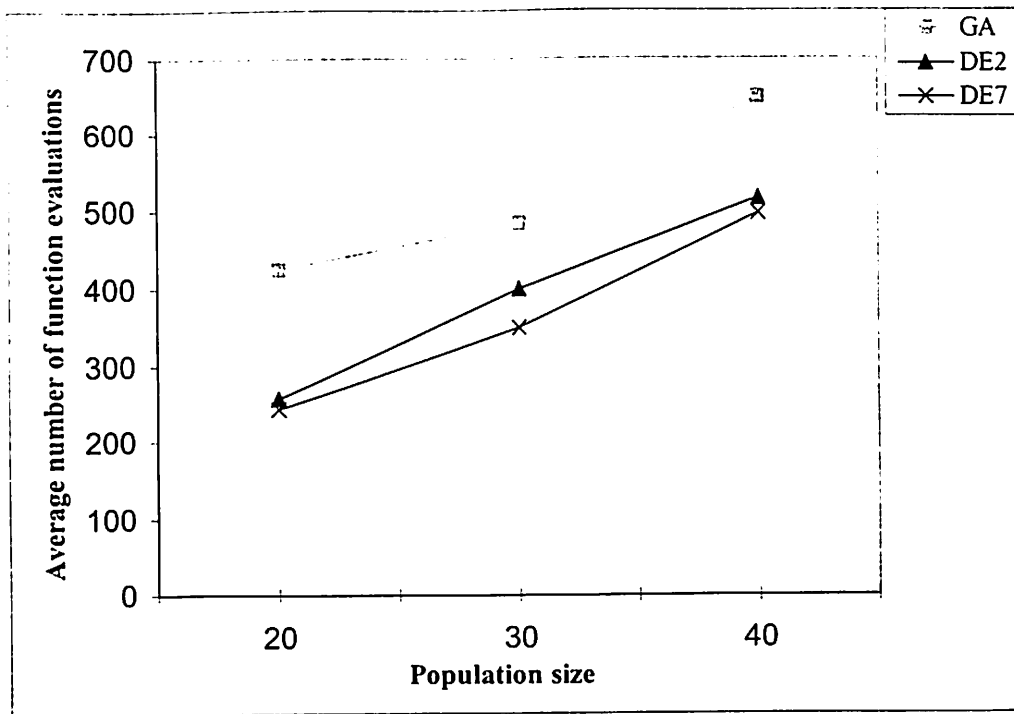


Fig. 2.2. Comparison of DE and GA

## 2.5. Conclusions

In this chapter, the optimization of a non-linear function (Himmelblau function) using Differential Evolution (an evolutionary computation method) is presented. The strategy no. 7 (that takes minimum number of function evaluations) is found to be the best for the present problem. From the results, it is clear that  $NFE_{DE} < NFE_{GA}$  and  $NRC_{DE} > NRC_{GA}$ . Hence it can be concluded that the performance of DE is better than that of GA.

This successful application of DE over GA for the Himmelblau function indicates that DE has great potential and can be applied to advantage in all the highly non-linear & complex engineering problems. In the next chapter the performance of DE is compared with that of Modified differential evolution (MDE), which is proposed in this study, using several benchmark test functions and selected chemical engineering processes.

**MDE AND ITS APPLICATION TO TEST  
FUNCTIONS & SELECTED NON-LINEAR  
CHEMICAL PROCESSES**

**3.1. Introduction**

Differential Evolution (DE), one of the evolutionary algorithms, is a novel optimization method capable of handling nondifferentiable, nonlinear and multimodal objective functions. Previous studies have shown that differential evolution is an efficient, effective and robust evolutionary optimization method. Still, DE takes large computational time for optimizing the computationally expensive objective functions. And therefore, an attempt to speed up DE is considered necessary. This chapter introduces a modification to original DE that enhances the convergence rate without compromising on solution quality. Our Modified Differential Evolution (MDE) algorithm utilizes only one set of population as against two sets in original DE at any given point of time in a generation. Such an improvement reduces the memory and computational efforts. The proposed MDE is applied to benchmark test functions and selected non-linear chemical engineering problems.

### 3.2. Background

When using any population based search algorithm in general and DE in particular to optimize a function, an acceptable trade –off between convergence rate (with reference to locating optimum) and robustness (with reference to not missing the global optima) must generally be determined. Convergence rate implies a fast convergence although it may be to a local optimum. On the other hand, robustness guarantees a high probability of obtaining the global optimum. One simple method, to increase the convergence rate is to reduce the population size but doing so means population diversity is poor. When the population diversity is small, the candidate individuals can be closely clustered. Therefore, the mutation and crossover operations can no longer generate the next better individual. This fact leads to the premature convergence and hence poor robustness due to the diminishing of the difference vector.

A few attempts have already been made to achieve this trade off (Wang and Chiou, 1997; Chiou and Wang, 1999; Fan and Lampinen, 2003; Tasoulis et al., 2004; Bergey and Ragsdale, 2005). Wang and Chiou (1997) suggested adjustment of the scaling factor  $F$  of the mutation operator and the number of individuals  $NP$  in DE. If  $NP$  is increased while simultaneously slightly lowering its  $F$  value, DE becomes increasingly robust. However, by doing so, much computation time should be expended to evaluate the objective function. This fact is particularly relevant when using DE to solve optimal control problems due to the large amount of CPU time required for solving the differential equations.

Chiou and Wang (1999) embedded accelerated phase and migration phase into the original algorithm of DE. These two phases are used to improve the convergence speed without decreasing the diversity among individuals. Also, several alternate



strategies are compared. Fan and Lampinen (2003) proposed a trigonometric mutation to enhance the convergence velocity of DE. They concluded that the convergence rate of algorithm could be significantly increased within a given maximum CPU time. Tasoulis et al. (2004) explored how differential evolution can be parallelized in a virtual parallel environment so as to improve both the speed and the performance of the method. Bergey and Ragsdale (2005) proposed a DE with greedy random strategy for genetic recombination. They found that modified algorithm has higher convergence velocity than original DE still maintaining the diversity.

This chapter discusses the attempt made in this study to increase the convergence speed of DE without compromising with the robustness (possibility of obtaining the global optimum). A Modified Differential Evolution (MDE) is proposed in the present work to achieve this trade – off.

### **3.3. Modified Differential Evolution (MDE)**

The principle of modified DE is same as DE. The schematic diagrams of DE & MDE are shown in Figs. 3.1 & 3.2 respectively. The major difference between DE and MDE is that MDE maintains only one array as is evident from both the figures. The array is updated as and when a better solution is found. Also, these newly found better solutions can take part in mutation and crossover operation in the current generation itself as opposed to DE (where another array is maintained and these better solutions take part in mutation and crossover operations in next generation). Updating the single array continuously enhances the convergence speed leading to less function evaluations as compared to DE. However, DE maintains two arrays consuming extra memory and CPU-time (more function evaluations). The schematic diagram of MDE algorithm (Fig. 3.2) clearly shows the use of single array in MDE instead of double

array in DE. This modification enables the algorithm to get a better trade-off between the convergence rate and the robustness.

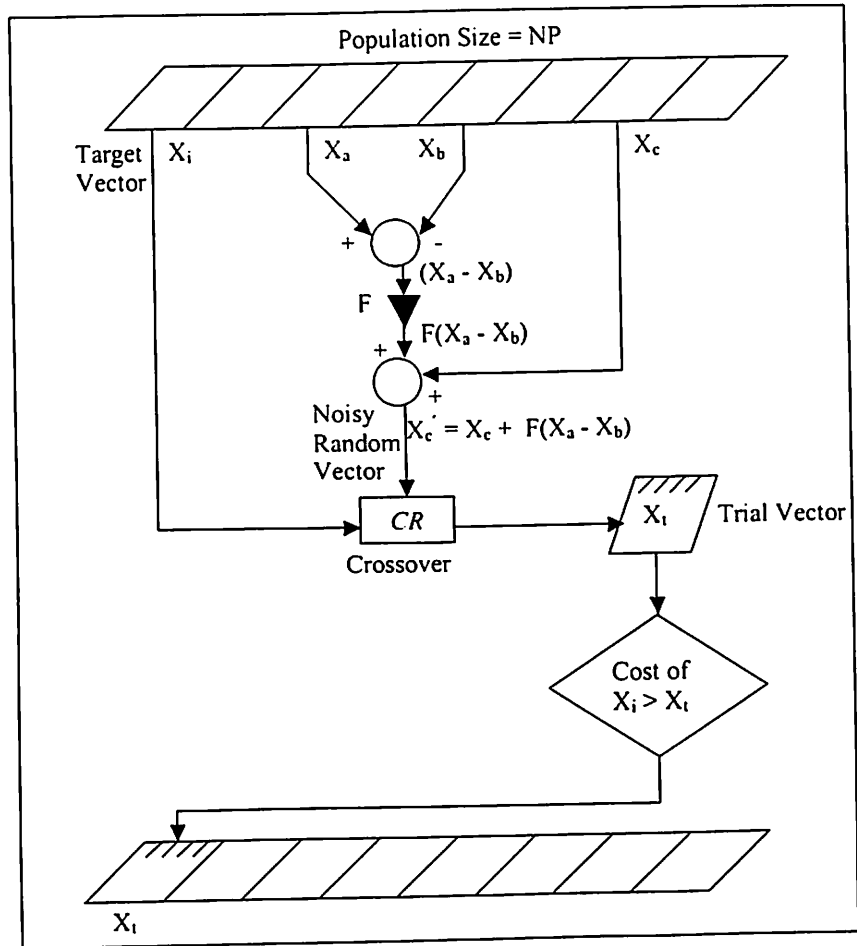


Fig. 3.1. Schematic of DE

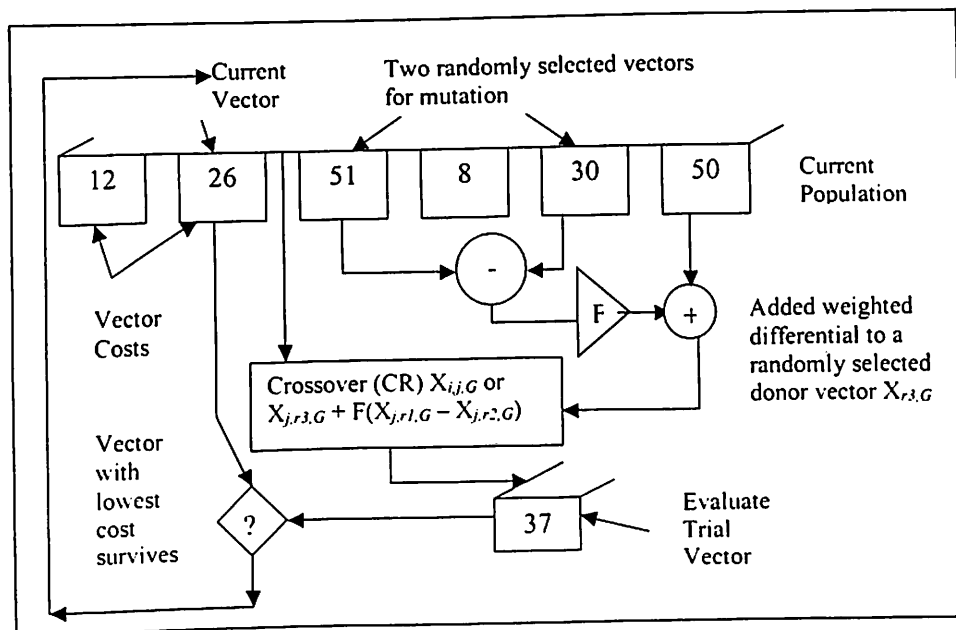


Fig. 3.2. Schematic of MDE

Thus it is possible to increase the convergence rate of the differential evolution algorithm and thereby obtain an acceptable solution with a lower number of objective function evaluations. Such an improvement can be advantageous in many real-world problems where the evaluation of a candidate solution is a computationally expensive operation and consequently finding the global optimum or a good sub-optimal solution with the original differential evolution algorithm is too time-consuming, or even impossible within the time available. The pseudo code of the proposed MDE is given below:

*Let  $P$  a population of size  $NP$ ,*

*and  $\bar{x}^j$  the  $j^{\text{th}}$  individual of dimension  $D$  in population  $P$ ,*

*and  $CR$  denotes the crossover probability*

**input**  $D, NP \geq 4; F \in (0, 1+); CR \in [0, 1]$ , and initial

*bounds:  $lower(x_i); upper(x_i); i = 1, \dots, D$*

*initialize  $P = \{ \bar{x}^1, \dots, \bar{x}^{NP} \}$  as*

**For each individual  $j \in P$**

$$\bar{x}_i^j = lower(x_i) + rand_i [0, 1] \times (upper(x_i) - lower(x_i)); i = 1, \dots, D$$

**end For each**

*Evaluate  $P$*

**while the stopping criterion is not satisfied do**

**forall  $j \leq NP$**

*Randomly select  $r_1, r_2, r_3 \in (1, \dots, NP)$ ,*

*$j \neq r_1 \neq r_2 \neq r_3$*

*randomly select  $i_{rand} \in (1, \dots, D)$*

**forall  $i \leq D$ ,**

$$x_i^j = \begin{cases} x_i^{r_3} + F \times (x_i^{r_1} - x_i^{r_2}) & \text{if } (random [0,1] < CR \wedge i = i_{rand}) \\ x_i^j & \text{otherwise} \end{cases}$$

**end forall**

if  $f(\bar{x}') \leq f(\bar{x}^j)$

Then,  $\bar{x}^j = \bar{x}'$ ;  $f(\bar{x}^j) = f(\bar{x}')$ ;

end forall

end while

Print the results.

### 3.3.1. Effect of Control Parameters (CR and F)

To study the effect of control parameters (CR and F), two highly multimodal multidimensional test functions are used. These are discussed briefly below:

*Ackley's function* (Bäck, 1996; Fan and Lampinen, 2003): This is a continuous, highly nonlinear function that causes the search with moderate complications.

$$f_1 = -20 \cdot \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)\right) + 20 + e;$$

$$-20 \leq x_i \leq 30, n = 30.$$

The global minimum is:  $f_1 = 0$  with  $x_i = 0, i = 1, 2, \dots, n$ .

*Rastrigin's function* (Muhlenbein et al., 1991): This function is also considered relatively difficult to minimize because the number of locally optimum points is high.

$$f_2 = 2n + \sum_{i=1}^n (x_i^2 - 2 \cos(2\pi x_i));$$

$$-5.12 \leq x_i \leq 5.12, n = 20.$$

The global minimum is:  $f_2 = 0$  with  $x_i = 0, i = 1, 2, \dots, n$ .

Fig. 3.3 and Fig. 3.4 show the effect of CR and F respectively on the performance of MDE and DE using the two functions as mentioned above. Each point on the figure represents the average of 100 experiments. It is evident from these figures that both DE and MDE are affected in a similar way, i.e., the variation of function value with CR and F is same qualitatively. It is important to note that variation of function value with CR and F is different for the same problem but it is same for DE and MDE.

if  $f(\bar{x}') \leq f(\bar{x}^j)$

Then,  $\bar{x}^j = \bar{x}'$ ;  $f(\bar{x}^j) = f(\bar{x}')$ ;

end forall

end while

Print the results.

### 3.3.1. Effect of Control Parameters (CR and F)

To study the effect of control parameters (CR and F), two highly multimodal multidimensional test functions are used. These are discussed briefly below:

*Ackley's function* (Bäck, 1996; Fan and Lampinen, 2003): This is a continuous, highly nonlinear function that causes the search with moderate complications.

$$f_1 = -20 \cdot \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)\right) + 20 + e;$$

$$-20 \leq x_i \leq 30, n = 30.$$

The global minimum is:  $f_1 = 0$  with  $x_i = 0, i = 1, 2, \dots, n$ .

*Rastrigin's function* (Muhlenbein et al., 1991): This function is also considered relatively difficult to minimize because the number of locally optimum points is high.

$$f_2 = 2n + \sum_{i=1}^n (x_i^2 - 2 \cos(2\pi x_i));$$

$$-5.12 \leq x_i \leq 5.12, n = 20.$$

The global minimum is:  $f_2 = 0$  with  $x_i = 0, i = 1, 2, \dots, n$ .

Fig. 3.3 and Fig. 3.4 show the effect of CR and F respectively on the performance of MDE and DE using the two functions as mentioned above. Each point on the figure represents the average of 100 experiments. It is evident from these figures that both DE and MDE are affected in a similar way, i.e., the variation of function value with CR and F is same qualitatively. It is important to note that variation of function value with CR and F is different for the same problem but it is same for DE and MDE.

Also, the variation of function value with  $CR$  and  $F$  is problem dependent, i.e., different problems can have different trends of function value vs.  $CR/F$ . Therefore, for comparison purposes, it is essential to keep the same setting of control parameters for both DE and MDE although this setting can be different for different problems.

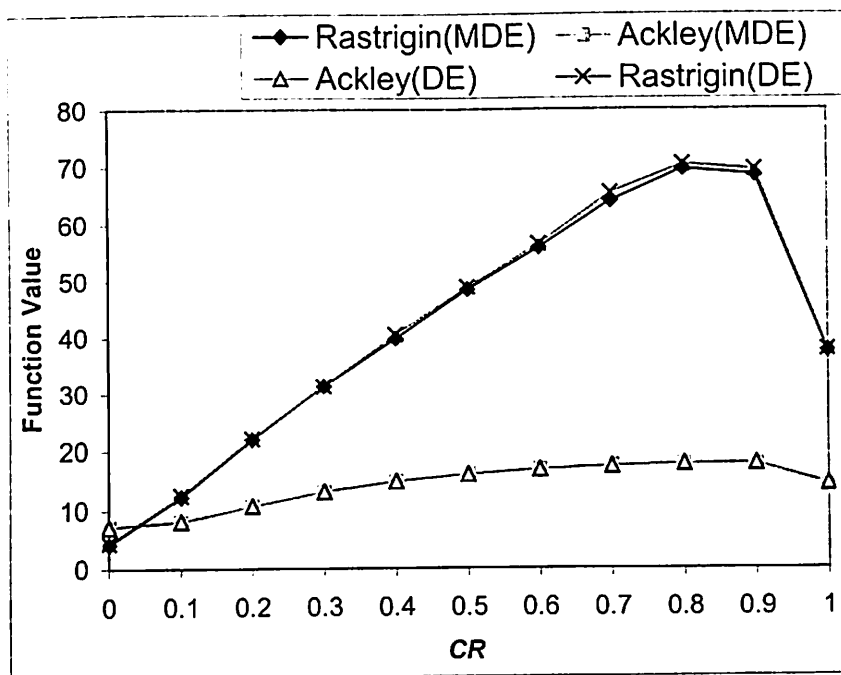


Fig. 3.3. Effect of  $CR$

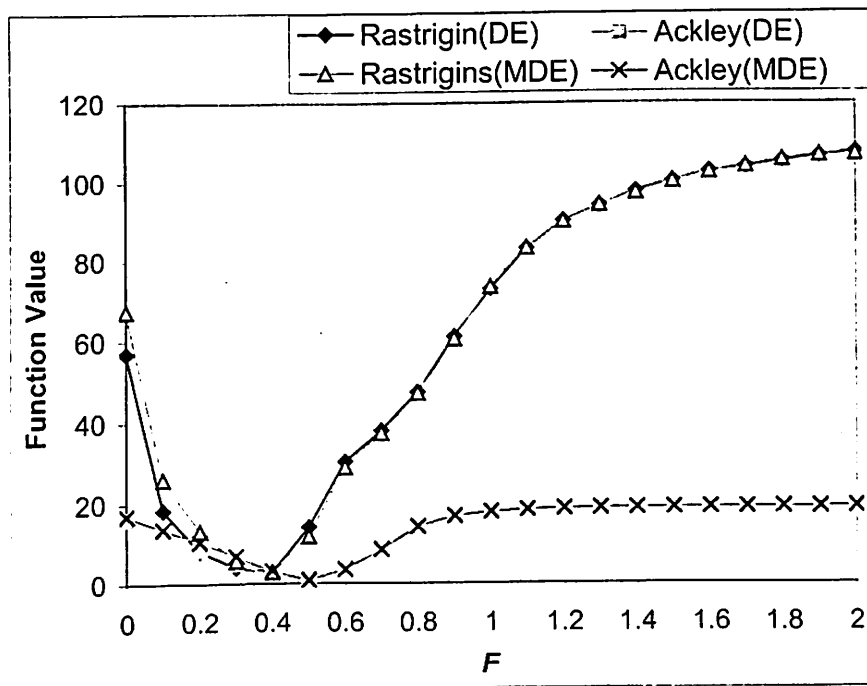


Fig. 3.4. Effect of  $F$

The MDE algorithm is demonstrated by applying them to several benchmark test functions, and is further examined with selected chemical engineering problems. The obtained numerical simulation results are providing empirical evidences on the efficiency and effectiveness of the proposed modified differential evolution algorithm.

In this chapter, performance of the algorithm, proposed in the present study (Angira and Babu, 2005b; Babu and Angira, 2005b), i.e., MDE is compared with the performance of DE by applying both these algorithms to some benchmark test functions reported in literature (**HIM**: Himmelblau function; **GP<sub>2</sub>**: Goldstein and Price function; **ES<sub>2</sub>**: Easom function; **H<sub>3</sub>**: Hartmann function; **R<sub>2</sub>, R<sub>5</sub> & R<sub>10</sub>**: Rosenbrock function; **Z<sub>2</sub>, Z<sub>5</sub> & Z<sub>10</sub>**: Zakharov function, subscript 2, 3, 5, and 10 indicate dimensions of the problems chosen). Further the reliability & efficiency of proposed algorithm (MDE) is evaluated on several selected non-linear chemical processes and the performance of MDE is compared with that of DE.

### **3.4. Application to Benchmark Test Functions**

To test the reliability & efficiency of proposed optimization technique viz., MDE several multimodal test functions are used. Various researchers (Bilbro and Snyder, 1991; Cvijovic and Klinowsk, 1995; Siarry and Berthiau, 1997; Chelouah and Siarry, 2000) had already used some of these functions to test their optimization methods. Recently, Teh and Rangaiah (2003) used these functions to check the reliability and efficiency of TS-QN (a hybrid method of Tabu search and Quasi-Newton). The selected functions are briefly described below and details of the global minimum are summarized in Table-3.1.

The MDE algorithm is demonstrated by applying them to several benchmark test functions, and is further examined with selected chemical engineering problems. The obtained numerical simulation results are providing empirical evidences on the efficiency and effectiveness of the proposed modified differential evolution algorithm.

In this chapter, performance of the algorithm, proposed in the present study (Angira and Babu, 2005b; Babu and Angira, 2005b), i.e., MDE is compared with the performance of DE by applying both these algorithms to some benchmark test functions reported in literature (**HIM**: Himmelblau function; **GP<sub>2</sub>**: Goldstein and Price function; **ES<sub>2</sub>**: Easom function; **H<sub>3</sub>**: Hartmann function; **R<sub>2</sub>, R<sub>5</sub> & R<sub>10</sub>**: Rosenbrock function; **Z<sub>2</sub>, Z<sub>5</sub> & Z<sub>10</sub>**: Zakharov function, subscript 2, 3, 5, and 10 indicate dimensions of the problems chosen). Further the reliability & efficiency of proposed algorithm (MDE) is evaluated on several selected non-linear chemical processes and the performance of MDE is compared with that of DE.

### **3.4. Application to Benchmark Test Functions**

To test the reliability & efficiency of proposed optimization technique viz., MDE several multimodal test functions are used. Various researchers (Bilbro and Snyder, 1991; Cvijovic and Klinowsk, 1995; Siarry and Berthiau, 1997; Chelouah and Siarry, 2000) had already used some of these functions to test their optimization methods. Recently, Teh and Rangaiah (2003) used these functions to check the reliability and efficiency of TS-QN (a hybrid method of Tabu search and Quasi-Newton). The selected functions are briefly described below and details of the global minimum are summarized in Table-3.1.



### Problem-1

Minimize **HIM**.

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

Subject to  $0 \leq x_1, x_2 \leq 6$ .

### Problem-2

Minimize **GP<sub>2</sub>**.

$$f(x) = [1 + (A1 * A2)][30 + (A3 * A4)]$$

$$\text{Where } A1 = (x_1 + x_2 + 1)^2;$$

$$A2 = (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2);$$

$$A3 = (2x_1 - 3x_2^2);$$

$$A4 = (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2);$$

Subject to  $-2 \leq x_1, x_2 \leq 2$ .

### Problem-3

Minimize **ES<sub>2</sub>**.

$$f(x) = -\cos(x_1)\cos(x_2)\exp[-((x_1 - \pi)^2 + (x_2 - \pi)^2)]$$

Subject to  $-100 \leq x_1, x_2 \leq 100$ .

### Problem-4

Minimize **H<sub>3</sub>**.

$$\text{Min } f(x) = -\sum_{i=1}^4 c_i \exp\left[-\sum_{j=1}^3 a_{ij}(x_j - p_{ij})^2\right]$$

Subject to  $0 \leq x_j \leq 1, j = 1, 2, 3$ ;

$$a_{ij} = \begin{bmatrix} 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \\ 3.0 & 10.0 & 30.0 \\ 0.1 & 10.0 & 35.0 \end{bmatrix}; \quad c_i = \begin{bmatrix} 1.0 \\ 1.2 \\ 3.0 \\ 3.2 \end{bmatrix}; \quad p_{ij} = \begin{bmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.0381 & 0.5743 & 0.8828 \end{bmatrix}$$

### Problem-5, 6, and 7

Problem-5, 6, and 7 corresponds to dimension of 2, 5, and 10 respectively.

Minimize **R<sub>D</sub>**.

$$f(x) = \sum_{i=1}^{D-1} [100(x_i^2 - x_{i-1})^2] + [x_D - 1]^2$$

Subject to  $-5 \leq x_i \leq 10; \quad i = 1, \dots, D.$

### Problem-8, 9, and 10

Problem-8, 9, and 10 corresponds to dimension of 2, 5, and 10 respectively.

Minimize  $Z_D$ .

$$f(x) = \left( \sum_{i=1}^D x_i^2 \right) + \left( \sum_{i=1}^D 0.5ix_i \right)^2 + \left( \sum_{i=1}^D 0.5ix_i \right)^4$$

Subject to  $-5 \leq x_i \leq 10, i = 1, \dots, D.$

**Table 3.1. Details of global minimum**

Function	$D$	Global minimum	Remarks
HIM	2	0 at $x = \{3, 2\}$	Three local minima
GP <sub>2</sub>	2	3 at $x = \{0, -1\}$	Four local minima
ES <sub>2</sub>	2	-1 at $x = \{\pi, \pi\}$	Several local minima
H <sub>3</sub>	3	-3.86278 at $x = \{0.114624, 0.555649, 0.852547\}$	Four local minima
R <sub>D</sub>	2, 5, and 10	0 at $x = \{1, \dots, 1\}$	Several local minima
Z <sub>D</sub>	2, 5, and 10	0 at $x = \{0, \dots, 0\}$	Several local minima

#### 3.4.1. Results and Discussion

Table-3.2 shows the results obtained using DE and MDE. Both the methods are coded in C language using Borland C. The results are compared on the basis of average CPU-time and success rate in locating the global optimum for the given tolerance, in an overall ten executions implemented. The termination criterion used is  $|OF_{cal} - OF_{Anal}| < 1 \times 10^{-6}$  for both DE & MDE for comparison purposes. It is clear that MDE takes less CPU-time and hence number of function evaluations. Since MDE uses only one array for population therefore it uses less computer memory as compared to DE (which uses two arrays for population). All the executions are made

$$f(x) = \sum_{i=1}^{D-1} [100(x_i^2 - x_{i+1})^2] + [x_i - 1]^2$$

Subject to  $-5 \leq x_i \leq 10; \quad i = 1, \dots, D.$

### Problem-8, 9, and 10

Problem-8, 9, and 10 corresponds to dimension of 2, 5, and 10 respectively.

Minimize  $Z_D$ .

$$f(x) = \left( \sum_{i=1}^D x_i^2 \right) + \left( \sum_{i=1}^D 0.5ix_i \right)^2 + \left( \sum_{i=1}^D 0.5ix_i \right)^4$$

Subject to  $-5 \leq x_i \leq 10, i = 1, \dots, D.$

**Table 3.1. Details of global minimum**

Function	$D$	Global minimum	Remarks
HIM	2	0 at $x = \{3, 2\}$	Three local minima
GP <sub>2</sub>	2	3 at $x = \{0, -1\}$	Four local minima
ES <sub>2</sub>	2	-1 at $x = \{\pi, \pi\}$	Several local minima
H <sub>3</sub>	3	-3.86278 at $x = \{0.114624, 0.555649, 0.852547\}$	Four local minima
R <sub>D</sub>	2, 5, and 10	0 at $x = \{1, \dots, 1\}$	Several local minima
Z <sub>D</sub>	2, 5, and 10	0 at $x = \{0, \dots, 0\}$	Several local minima

#### 3.4.1. Results and Discussion

Table-3.2 shows the results obtained using DE and MDE. Both the methods are coded in C language using Borland C. The results are compared on the basis of average CPU-time and success rate in locating the global optimum for the given tolerance, in an overall ten executions implemented. The termination criterion used is  $|OF_{cal} - OF_{Anal}| < 1 \times 10^{-6}$  for both DE & MDE for comparison purposes. It is clear that MDE takes less CPU-time and hence number of function evaluations. Since MDE uses only one array for population therefore it uses less computer memory as compared to DE (which uses two arrays for population). All the executions are made

on the same machine with the same platform. The strategy used is DE/rand/1/bin in all the experiments.

**Table-3.2. Results of DE and MDE**

S. No.	NFE (CPU-time)		Percentage* Time saving	SR (%)	
	DE	MDE		DE	MDE
HIM	956 (0.000)	848 (0.000)	Nil	100	100
ES <sub>2</sub>	3052 (0.094)	2512 (0.083)	11.7	100	100
GP <sub>2</sub>	1222 (0.020)	1024 (0.016)	20.0	100	100
R <sub>2</sub>	2056 (0.055)	2042 (0.055)	Nil	100	100
Z <sub>2</sub>	716 (0.022)	704 (0.016)	27.27	100	100
H <sub>3</sub>	1704 (0.066)	1563 (0.055)	16.67	100	100
R <sub>5</sub>	50020 (2.033)	49525 (2.015)	0.88	100	100
Z <sub>5</sub>	10370 (0.418)	9525 (0.374)	10.53	100	100
R <sub>10</sub>	417510 (26.962)	417280 (26.758)	0.76	100	100
Z <sub>10</sub>	139530 (8.654)	136530 (8.159)	5.72	100	100

\* % Time saving =  $100 \times [(CPU-time)_{DE} - (CPU-time)_{MDE}] / (CPU-time)_{DE}$

There is saving in CPU-time in most of the test functions using MDE. The saving is significant in ES<sub>2</sub>, GP<sub>2</sub>, Z<sub>2</sub>, H<sub>3</sub>, and Z<sub>5</sub> as compared to HIM, R<sub>2</sub>, R<sub>5</sub>, and R<sub>10</sub> that can be explained using error analysis. Error is the difference between average cost of population and cost or objective function value corresponding to known global optimum in a generation.

Fig. 3.5 shows the error variation of DE and MDE for the test function named GP<sub>2</sub>. Error variation indicates that the error is more in case of MDE than DE till 3<sup>rd</sup> generation. After that it reduces at faster rate and becomes much less than that in DE (at 22<sup>nd</sup> generation, error using DE is 11.22 while using MDE it is 0.841). This clearly explains the higher speed of MDE to attain global optimum.

Also, it is evident from Fig. 3.6, 3.7, 3.8, and 3.9 that there is continuous decrease of error with generation using both DE and MDE. But error is reducing fast in case of MDE as compared to DE. This explains the fact that MDE is able to locate global optimum faster than DE.

It is to be noted that in case of test problem  $R_{10}$  (Fig. 3.10), error is nearly same as that of DE indicating almost same computation time to reach to global optimum. The results obtained (Table-3.2) substantiate it, since percent time saving is very less (about 0.76% only).

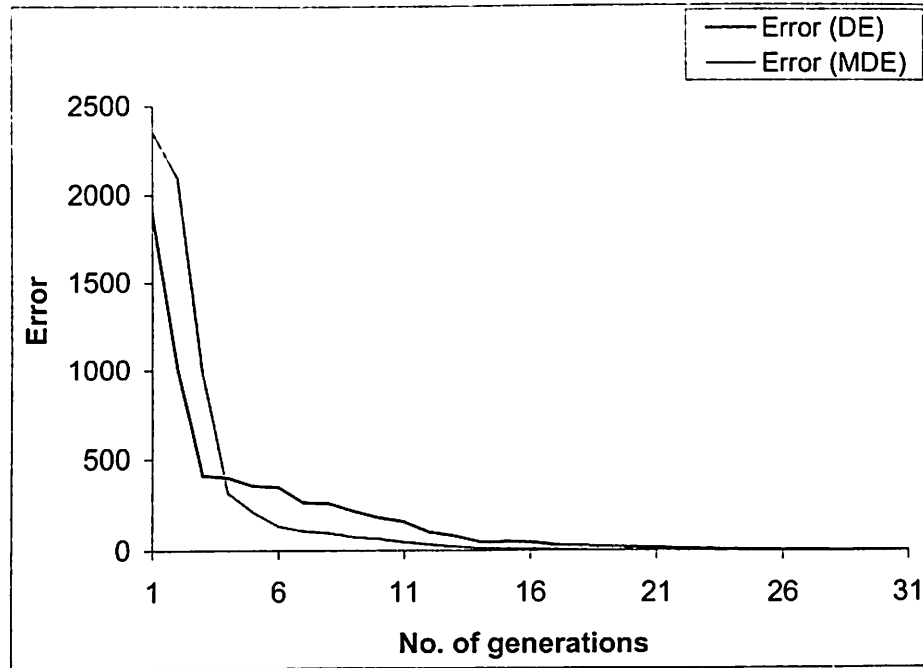


Fig. 3.5. Error variation for  $GP_2$

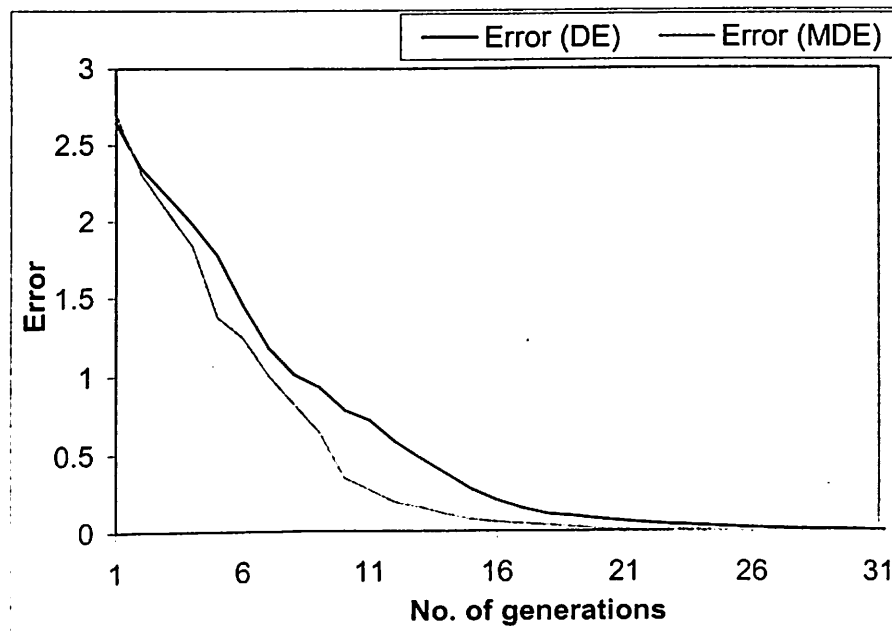


Fig. 3.6. Error variation for  $H_3$

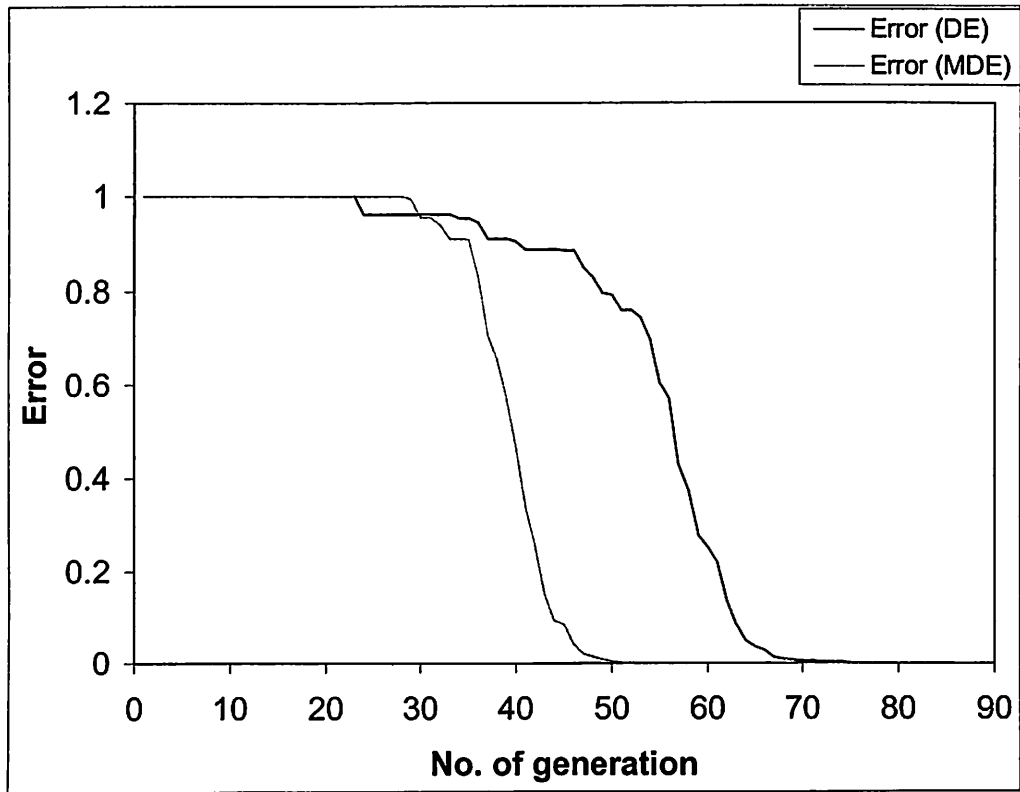


Fig. 3.7. Error variation for ES<sub>2</sub>

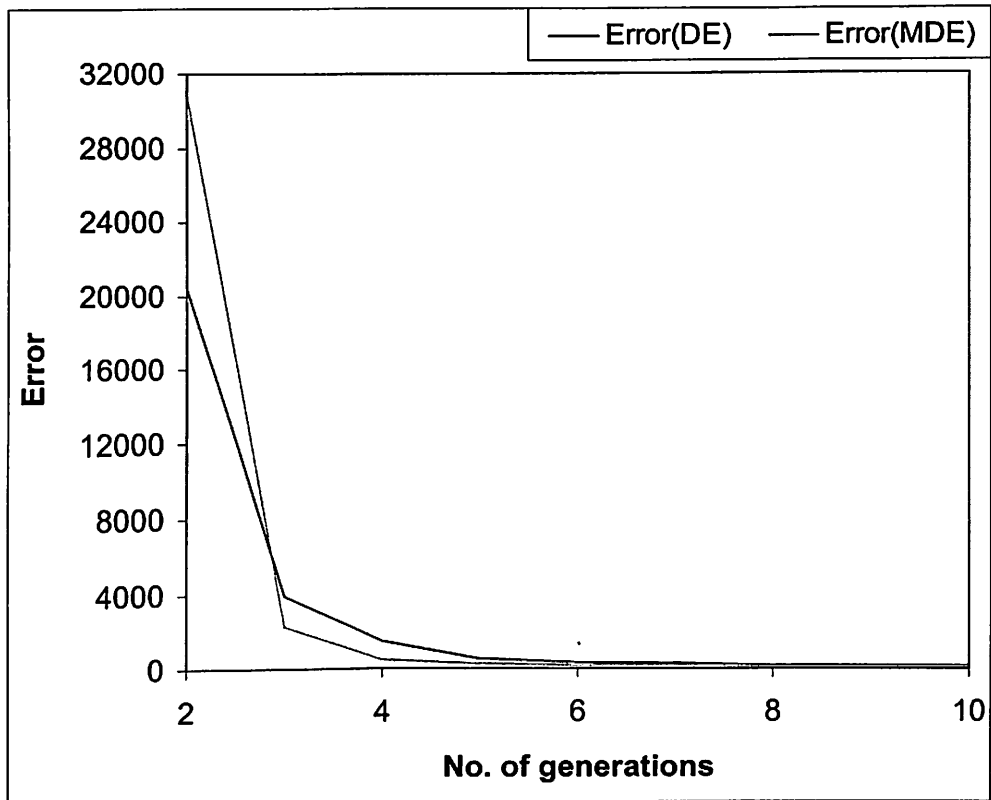


Fig. 3.8. Error variation for Z<sub>10</sub> ((2 to 10) x 5 generations)

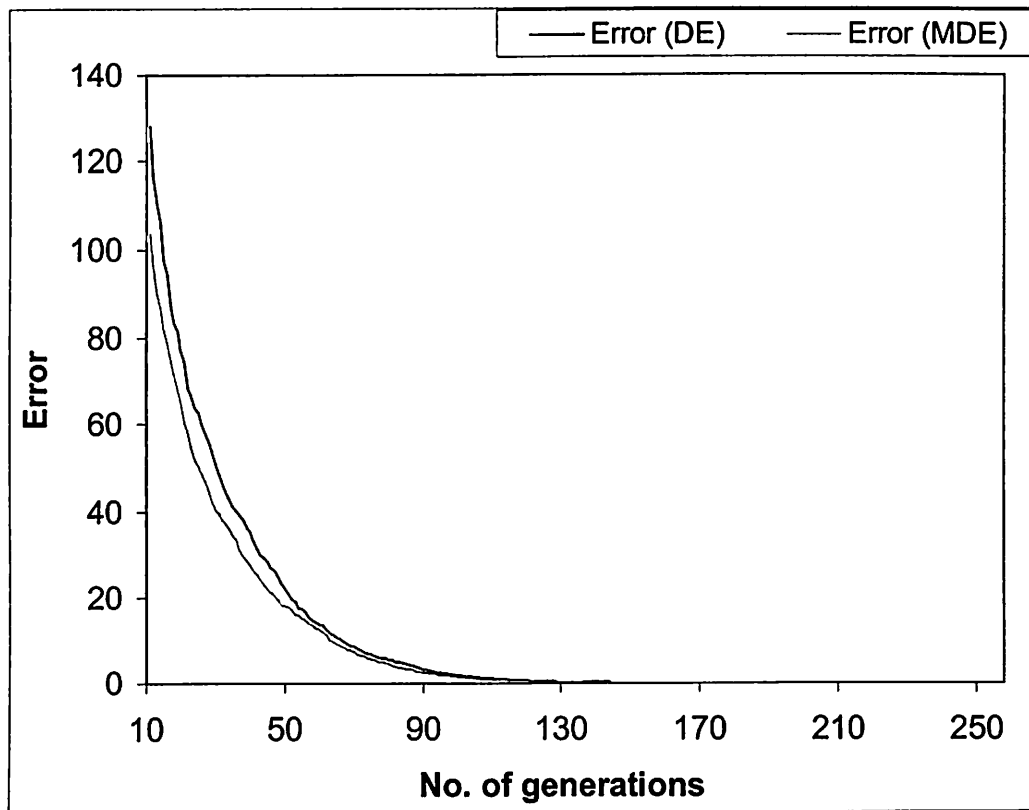


Fig. 3.9. Error variation for  $Z_{10}$  ((11 to 250) x 5 generations)

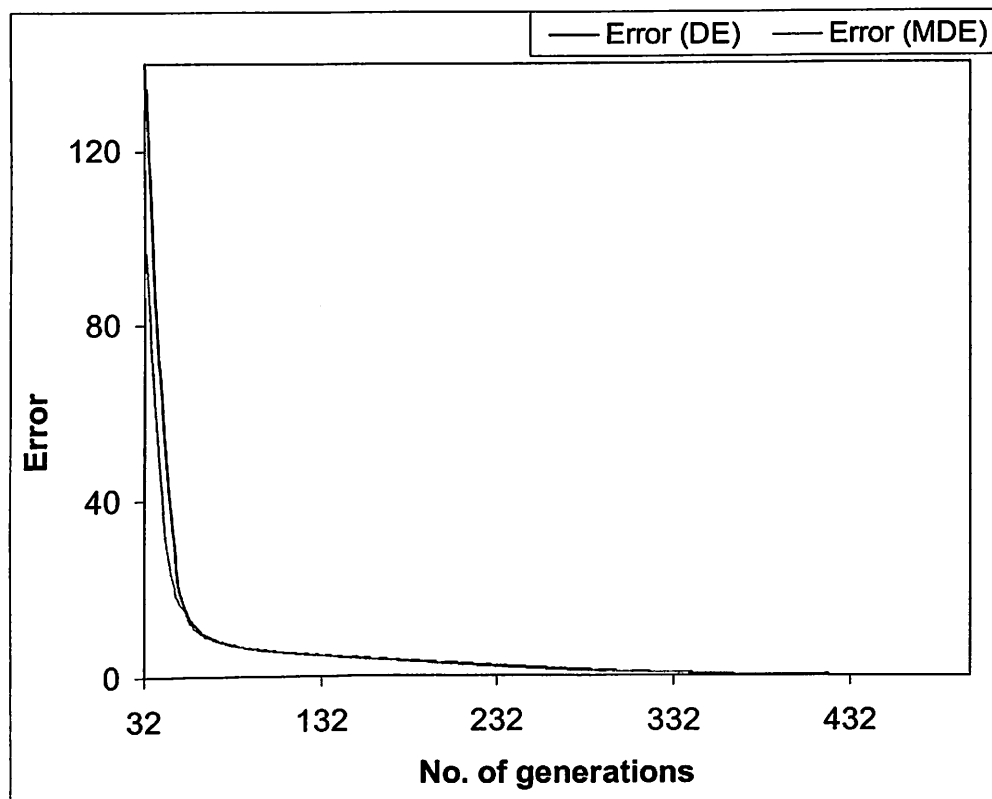


Fig. 3.10. Error variation for  $R_{10}$

### 3.4.2. Conclusions

It is found that the two algorithms (viz., DE, MDE) are reliable in locating the global optima of all the test problems. This is evident from success rate which is 100% for all the test problems using both DE and MDE. The performance of the newly proposed algorithm in this study, i.e., MDE is found to be better than that of DE. The results obtained by these methods (viz. DE, and MDE algorithm) are same and matches with that reported in literature.

The results stated above clearly show the potential of MDE and DE algorithms. In the next section, applying DE & MDE to nonlinear and constrained problems encountered in chemical engineering will test the performance of these algorithms.

### 3.5. Application to Selected ~~Non-Linear~~ Chemical Processes

Realistic treatments of physical and engineering systems frequently involve nonlinear models. Non-linearities are introduced by process equipment design relations, equilibrium relations and combined heat & mass balances. The design variables may be continuous [non-linear programming (NLP) problems] or mixed type consisting of floating and integer [mixed integer non-linear programming (MINLP) problems]. Model nonlinearities give rise to nonconvexities, which in turn, lead to multiple local optima. Gradient optimization techniques have only been able to tackle special formulations, where continuity or convexity had to be imposed, or by exploiting special mathematical structures. Stochastic algorithms, also known as adaptive random search methods, have successfully tackled nonconvex problems, mostly in the area of chemical engineering.

The optimization of non-linear constrained problems is relevant to chemical engineering practice (Salcedo, 1992; Floudas, 1995). As discussed in section-3.4, DE



& MDE are found to give very good results for many benchmark test problems. In this section, application of DE and MDE is discussed which are applied to solve the selected linear and nonlinear chemical engineering problems; (1) optimization of thermal cracker operation (2) optimal design of ammonia synthesis reactor (3) reactor network design (4) isothermal CSTR design (5) optimal operation of alkylation unit (6) fuel allocation in power plant (7) drying process (8) water pumping system (9) liquid extraction problem, and (10) heat exchanger network design. Many of these problems are difficult non-linear optimization problems, with equality and inequality constraints. The performance of DE is compared with MDE and also with the methods used to solve as mentioned in literature.

### **3.5.1. Optimization Of Thermal Cracker Operation**

#### **3.5.1.1. Cracking**

Cracking is basically the heating of higher boiling petroleum fractions like heavy fuel oil at high temperature (above decomposition temperature) and pressure to produce lower boiling fractions such as ethylene, propylene, and gasoline etc. It is an endothermic reaction.

Cracking of heavier fuel oils is carried out to produce mainly high quality (having high octane number) petrol. Also, cracking is carried out to produce olefins which are used as feed for petrochemical industry, to produce coke (by coking) and to reduce the viscosity of fuel oil (by visbreaking). There are two types of cracking namely (i) Thermal cracking and (ii) Catalytic cracking. Fig. 3.11 shows the further classification of various cracking process that are used widely in chemical process industries. The details on the above eight types of cracking processes are available in literature (Hobson, 1975; Sourander et al, 1984; Gupta, 1994). Thermal cracking,

which is of interest in the present study (Babu and Angira, 2001b) is discussed below in detail.

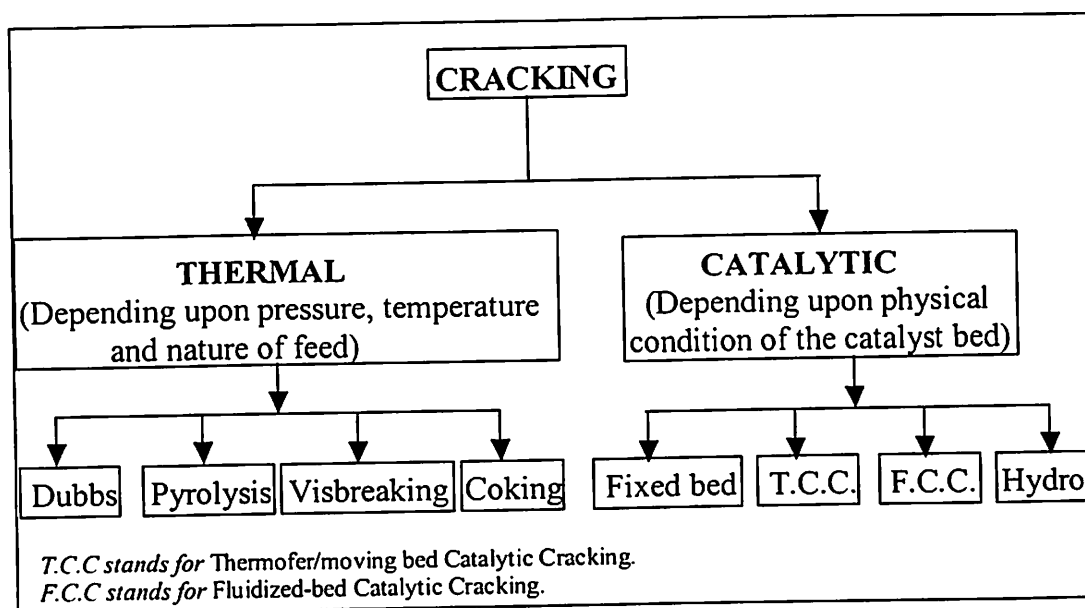


Fig. 3.11. Classification of cracking process

### 3.5.1.2. Thermal Cracking

It is defined as thermal decomposition, under pressure, of large hydrocarbon molecules to form smaller molecules. Lighter, more valuable hydrocarbons may thus be obtained from such relatively low value stocks as heavy fuel/gas oils (boiling up to 540°C) and residues. This is conducted without any catalyst. Thermal cracking is normally carried out at temperatures varying from 450°C to 750°C and pressures ranging from atmospheric to 1000 psig (Hobson, 1975). The important reactions occurring are:

- Decomposition and destructive condensation.
- Hydrogenation and dehydrogenation.
- Polymerization.
- Cyclization.

The first two reactions are endothermic, while polymerization is exothermic. Coke formation is an additional reaction which plays an important role in thermal cracking,

although the mechanism by which coke is formed is not entirely understood. It is thought, however, that coke results from extensive degradation of relatively heavy molecules to form increasing quantities of light hydrocarbon gases (dry gas) and polycyclic compounds having low hydrogen to carbon ratios. The rate, at which hydrocarbon cracks, is strongly dependent on temperature. Cracking reactions begin at about 315-370<sup>0</sup>C, depending on the type of material being cracked (Hobson, 1975).

Depending upon the pressure and temperature employed for the cracking and the characteristics of feed, there are various thermal cracking processes in which the product yields and characteristics are different (Gupta, 1994). Mainly there are four commercial processes employed for thermal cracking in oil refineries. They are:

- Dubbs thermal cracking process
- Pyrolysis.
- Visbreaking.
- Coking.

In the present study, the problem of optimization of thermal cracker (pyrolysis) operation is discussed.

### **3.5.1.3. Pyrolysis**

Pyrolysis is done mainly for the production of lighter products predominantly unsaturated such as olefins (ethylene, propylene) and naphthene polymers, diolefins, benzene & toluene etc (Sourander et al, 1984). It is carried out at high temperature (650-700<sup>0</sup>C) and low pressure. Pyrolysis is also a reaction that occurs in one of the zones of biomass gasification process. Babu and Chaurasia (2003a, 2003b, 2004a, 2004b, 2004c, 2004d) carried out extensive studies on various aspects of pyrolysis.

The main objective in thermal-cracker optimization is the estimation of the optimal flow rates of different feeds (viz. Gas-oil, Propane, Ethane & Debutanized

natural gasoline) to the cracking furnace under the restriction on ethylene and propylene production (Sourander et al, 1984). Thousands of combinations of feeds are possible. Hence the optimization needs an efficient strategy in searching for the global optimal solution.

### 3.5.1.4. Description of The Problem

This problem (Edgar & Himmelblau, 1989) deals with maximization of profit while operating within furnace and down stream process equipment constraints. Fig. 3.12 lists various feeds & corresponding product distribution for a thermal cracker which produces olefins.

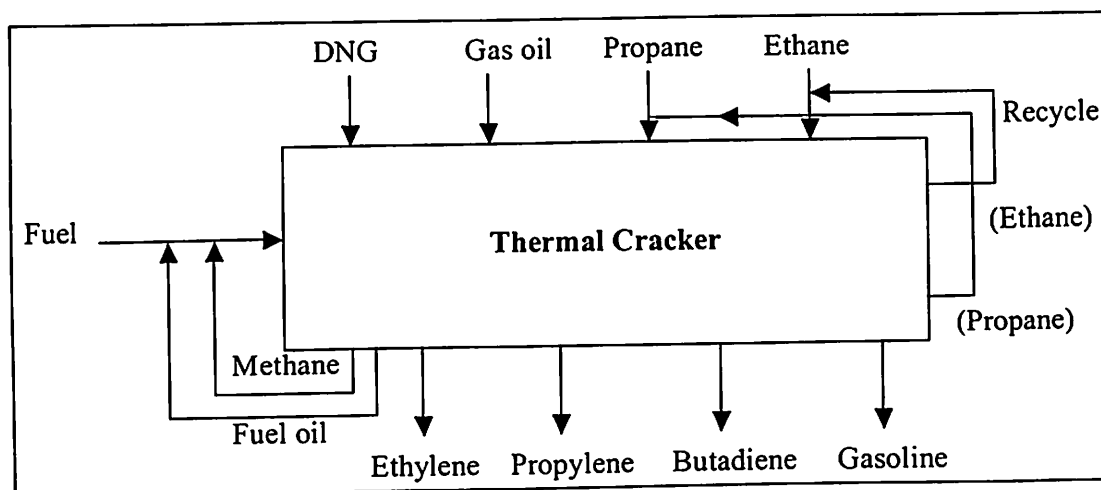


Fig. 3.12. Thermal cracker

Based on the plant data, eight products are produced in varying proportions according to the matrix shown in Table-3.3. The capacity to run gas feeds through the cracker is 200,000lb (90718.48 kg)/stream hr (total flow based on an average mixture). Ethane uses the equivalent of 1.1 lb of capacity per pound of ethane (1.1 kg/kg); propane uses 0.9 lb of capacity per pound of propane (0.9 kg/kg); gas-oil uses 0.9 lb/lb (0.9 kg/kg); and DNG has a ratio of 1.0. Down stream processing limits exist of 50,000 lb (22679.62 kg)/stream hr on the ethylene and 20,000 lb (9071.848 kg)/stream hr on the propylene. The fuel requirements to run the cracking system for each feedstock type are shown in Table-3.4. Methane and fuel oil produced by the

cracker are recycled as fuel. All the ethane and propane produced is recycled as feed. Heating values of fuels are given in Table-3.5. Assume an energy (fuel) cost of  $\$2.5/10^6$  Btu (Rs. 113.74/ $10^9$ J).

**Table-3.3. Product matrix with yield structure: (wt. fraction)**

Product	Feed			
	Ethane	Propane	Gas-oil	DNG
Methane	0.07	0.25	0.10	0.15
Ethane	0.40	0.06	0.04	0.05
Ethylene	0.50	0.35	0.20	0.25
Propane	0.00	0.10	0.01	0.01
Propylene	0.01	0.15	0.15	0.18
Butadiene	0.01	0.02	0.04	0.05
Gasoline	0.01	0.07	0.25	0.30
Fuel Oil	0.00	0.00	0.21	0.01
<b>Total</b>	1.00	1.00	1.00	1.00

**Table-3.4. Fuel requirements for each feedstock type**

Feed	Fuel requirement	
	(Btu/lb)	(J/kg)
Ethane	8364	19454664
Propane	5016	11667216
Gas oil	3900	9071400
DNG	4553	10590278

**Table-3.5. Heating value of Fuels**

Fuels	Heating Value	
	(Btu/lb)	(J/kg)
Natural Gas	21520	50055520
Methane	21520	50055520
Fuel oil	18000	41868000

**Table-3.6. Cost of feed, products and fuel**

	Cost		
	Cents/lb	Rs/kg	
Feeds	Ethane	6.55	6.93
	Propane	9.73	10.3
	Gas-oil	12.50	13.23
	DNG	10.14	10.73
Products	Methane	5.38 (fuel value)	5.69
	Ethylene	17.75	18.78
	Propylene	13.79	14.59
	Butadiene	26.64	28.19
	Gasoline	9.93	10.51
	Fuel-oil	4.50 (fuel value)	4.76

Because of heat losses and the energy requirements for pyrolysis, a fixed fuel requirement of  $20.0 \times 10^6$  Btu ( $2.11 \times 10^{10}$  J)/stream hr. occurs. The price structure on the feeds & products and fuel costs are (all values are in cents per pound) given in Table-3.6.

The variables to be optimized are the amounts of the four feeds (viz. Gas-oil, Propane, Ethane, and Debutanized Natural Gasoline (DNG)). This problem is solved using Linear Programming Simplex method, DE, and MDE. The assumption used in formulating the objective function and constraints are:

1.  $20.0 \times 10^6$  Btu ( $2.11 \times 10^{10}$  J)/hr fixed fuel requirement (methane) to compensate for the heat-loss.
2. All propane and ethane are recycled with the feed, and all methane and fuel oil will be recycled as fuel.

Objective function for the profit is given by:

$$f = 2.84x_1 - 0.22x_2 - 3.33x_3 + 1.09x_4 + 9.39x_5 + 9.51x_6.$$

Where  $f$  = profit function (cents/ hr.)

$x_1$  = Fresh ethane feed (lb/hr.)

$x_2$  = Fresh propane feed (lb/hr.)

$x_3$  = Gas-oil feed (lb/hr.)

$x_4$  = DNG feed (lb/hr.)

$x_5$  = Ethane recycle (lb/hr.)

$x_6$  = Propane recycle (lb/hr.)

$x_7$  = Fuel added (lb/hr.)

*Constraints:*

- a). Cracker capacity of 200,000 lb/hr (90718.48 kg/hr),

$$1.1x_1 + 0.9x_2 + 0.9x_3 + 1.0x_4 + 1.1x_5 + 0.9x_6 \leq 200,000$$

b). Ethylene processing limitation of 50,000lb/hr (22679.62 kg/hr),

$$0.5x_1 + 0.35x_2 + 0.2x_3 + 0.25x_4 + 0.5x_5 + 0.35x_6 \leq 50,000$$

c). Propylene processing limitation of 20,000lb/hr (9071.848 kg/hr),

$$0.01x_1 + 0.15x_2 + 0.15x_3 + 0.18x_4 + 0.01x_5 + 0.15x_6 \leq 20,000.$$

d). Ethane recycle

$$0.4x_1 + 0.06x_2 + 0.04x_3 + 0.05x_4 - 0.6x_5 + 0.06x_6 = 0.$$

e). Propane recycle

$$0.1x_2 + 0.01x_3 + 0.01x_4 - 0.9x_6 = 0.$$

f). Heat Constraints

$$-6857.6x_1 + 364x_2 + 2032x_3 - 1145x_4 - 6857.6x_5 + 364x_6 + 21,520x_7 = 20,000,000.$$

In the next section, the LP simplex method is briefly discussed.

### 3.5.1.5. *L.P. Simplex Method*

The standard LP form includes  $m$  simultaneous linear equations in  $n$  unknown variables ( $m < n$ ). We divide the  $n$  variables into two sets: (1)  $(n-m)$  variables, to which we assign zero values; and (2) the remaining  $m$  variables, whose values are determined by solving the resulting  $m$  equations. If the  $m$  equations yield a unique solution, then the associated  $m$  variables are called basic variables and the remaining  $(n-m)$  zero variables are referred to as nonbasic variables. In this case, the resulting unique solution comprises of a basic solution. If all the variables assume nonnegative values, then the basic solution is feasible. Otherwise, it is infeasible.

Based on the given definition, the maximum number of possible basic solutions for  $m$  equations in  $n$  unknowns is:

$$\frac{n!}{(n-m)!m!}$$

*Optimality condition:*

The entering variable in a maximization (minimization) problem is the nonbasic variable having the most negative (positive) coefficient in the Z-row i.e. objective function row. Ties are broken arbitrarily. The optimum is reached at the iteration where all the Z-row coefficients of the nonbasic variables are nonnegative (nonpositive).

*Feasibility condition:*

For both the maximization and minimization problems, the leaving variable is the basic variable associated with the smallest nonnegative ratio. In this case also, ties are broken arbitrarily.

For LPs in which all the constraints are of the ( $\leq$ ) type (with nonnegative right-hand sides), the slacks offer a convenient starting basic feasible solution. A natural question then arises: How can we find a starting basic solution for models that involve ( $=$ ) and ( $\geq$ ) constraints? The most common procedure for starting LPs that do not have convenient slacks is to use artificial variables. These are variables that assume the role of slacks at the first iteration, only to be disposed of at a later iteration. Two methods are proposed for effecting the solution of this problem (1) the Big M method and (2) the Two Phase method. The details on the procedures and difference of these two methods are discussed in detail and documented in literature (Taha, 1997). However, Big M method is used in the present study for comparison purposes of results with the original source (Edgar & Himmelblau, 1989).

The steps of Simplex method are (Taha, 1997):

- Step 1.** Convert the problem into standard L.P. form.
- Step 2.** Determine a starting basic feasible solution.
- Step 3.** Select an entering variable using the optimality condition. Stop if there is no entering variable.



**Step 4.** Select a leaving variable using the feasibility conditions.

**Step 5.** Determine the new basic solution using the appropriate Gauss-Jordan Computations. Go to Step 2.

### **3.5.1.6. Problem Reformulation**

The problem is reformulated by eliminating the equality constraints. This reformulation helped in reducing the number of constraints and decision variables. The reformulated problem (containing four decision variables and three constraints instead of seven variables & six constraints) is as follows:

$$\text{Max. } f = 9.1x_1 + 1.88x_2 - 2.5879x_3 + 1.9886x_4.$$

Constraints:

(a). Cracker capacity of 200,000 lb/hr (90718.48 kg/hr),

$$16.5x_1 + 10.1x_2 + 8.861x_3 + 9.926x_4 \leq 1800000.$$

(b). Ethylene processing limitation of 50,000lb/hr (22679.62 kg/hr),

$$7.5x_1 + 4.0x_2 + 2.14x_3 + 2.665x_4 \leq 450000.$$

(c). Propylene processing limitation of 20,000lb/hr (9071.848 kg/hr),

$$0.15x_1 + 1.51x_2 + 1.3711x_3 + 1.6426x_4 \leq 180000.$$

### **3.5.1.7. Results and Discussion**

In the previous section, we have discussed the problem formulation and reformulation of Thermal Cracker operation. Most of the engineering optimization problems are constrained. The difficulty of using EAs in the constrained optimization is that the evolutionary operators used to manipulate the individuals of the population often produce solution which are unfeasible. There are many methods to handle it. The following subsection discusses the handling of constraint in the present study followed by results and discussion.

### 3.5.1.7.1. Constraint Handling in EAs

Bound violations (whether upper or lower) may occur after mutation step of DE and MDE. This can be repaired by one of the following methods: (1). If there is bound violation for a parameter, then assign the upper or lower bound value if upper or lower bound is violated (*forced bound*) (2). If there is bound violation for a parameter, then that parameter is again generated randomly between given lower and upper bound (*without forcing*) using the following equation:

$$\bar{x}_i^j = \text{lower}(x_i) + \text{rand}_i [0, 1] \times (\text{upper}(x_i) - \text{lower}(x_i)); i = 1 \dots D$$

where  $D$  is the number of decision variables. In this thesis, the first situation is called forced bound method (FBM) while second situation is called method without forcing the bound (MWFB). The penalty function methods are one of the most popular techniques in EAs to handle constraints. The techniques transform the constrained problem into an unconstrained problem by penalizing unfeasible solutions. In addition, the penalty function methods are easy to implement and considered efficient. In the present study, an absolute value of constraint violation is multiplied with a high penalty and added/subtracted to objective function depending upon the type of problem, i.e., minimization/maximization. In case of more than one constraint, all such absolute violations are first multiplied with high penalty and then added or subtracted from objective function value (for minimization or maximization problem respectively). This method is followed throughout the thesis unless mentioned specifically.

### 3.5.1.7.2. Results of LP and DE

The reformulated problem is solved using DE, MDE & LP Simplex method and Table-3.7 & Table-3.8 show the results obtained. Table-3.7 shows the results obtained by both DE & LP Simplex method. It may be noted that the maximum

possible amount of ethylene is produced. As the ethylene production constraint is relaxed, the objective function value increases (Fig. 3.13). Once the constraint is raised above 90,909.0909 lb/hr (41235.673 kg/hr), the objective function remains constant at 676018.1875 cents/hr (324488.73 Rs/hr).

**Table-3.7. Results of LP Simplex and DE**

Stream	Flow Rate (DE)		Flow rate (LP Simplex)		Flow rate (Edgar & Himmelblau, 1989)	
	(lb/hr.)	(kg/hr)	(lb/hr.)	(kg/hr)	(lb/hr.)	(kg/hr)
Fresh Ethane feed ( $x_1$ )	60,000	27215.54	60,000	27215.54	60,000	27215.54
Fresh propane feed ( $x_2$ )	0	0	0	0	0	0
Gas-oil feed ( $x_3$ )	0	0	0	0	0	0
DNG feed ( $x_4$ )	0	0	0	0	0	0
Ethane recycle ( $x_5$ )	40,000	18143.7	40,000	18143.7	40,000	18143.7
Propane recycle ( $x_6$ )	0	0	0	0	0	0
Fuel added ( $x_7$ )	32795.54	14875.807	32795.54	14875.807	32,800	14877.83
Ethylene	50,000	22679.62	50,000	22679.62	50,000	22679.62
Propylene	1000	453.5924	1000	453.5924	1000	453.5924
Butadiene	1000	453.5924	1000	453.5924	1000	453.5924
Gasoline	1000	453.5924	1000	453.5924	1000	453.5924
Methane	7000	3175.15	7000	3175.15	7000	3175.15
Fuel oil	0	0	0	0	0	0
Objective function (cents/hr.)	369560.0	177388.8	369560.0	177388.8	369560.0	177388.8

**Table-3.8. Results of DE (all ten strategies)**

S. No.	Strategy	NFE	CPU- time	NRC
1	DE/rand/1/bin	6268	0.28	100
2	DE/best/1/bin	3168	0.145	100
3	DE/best/2/bin	9076	0.418	100
4	DE/rand/2/bin	11696	0.539	100
5	DE/rand-to-best/1/bin	6052	0.28	100
6	DE/rand/1/exp	5252	0.22	100
7	DE/best/1/exp	2796	0.126	100
8	DE/best/2/exp	10132	0.44	100
9	DE/rand/2/exp	12600	0.55	100
10	DE/rand-to-best/1/exp	6536	0.275	100

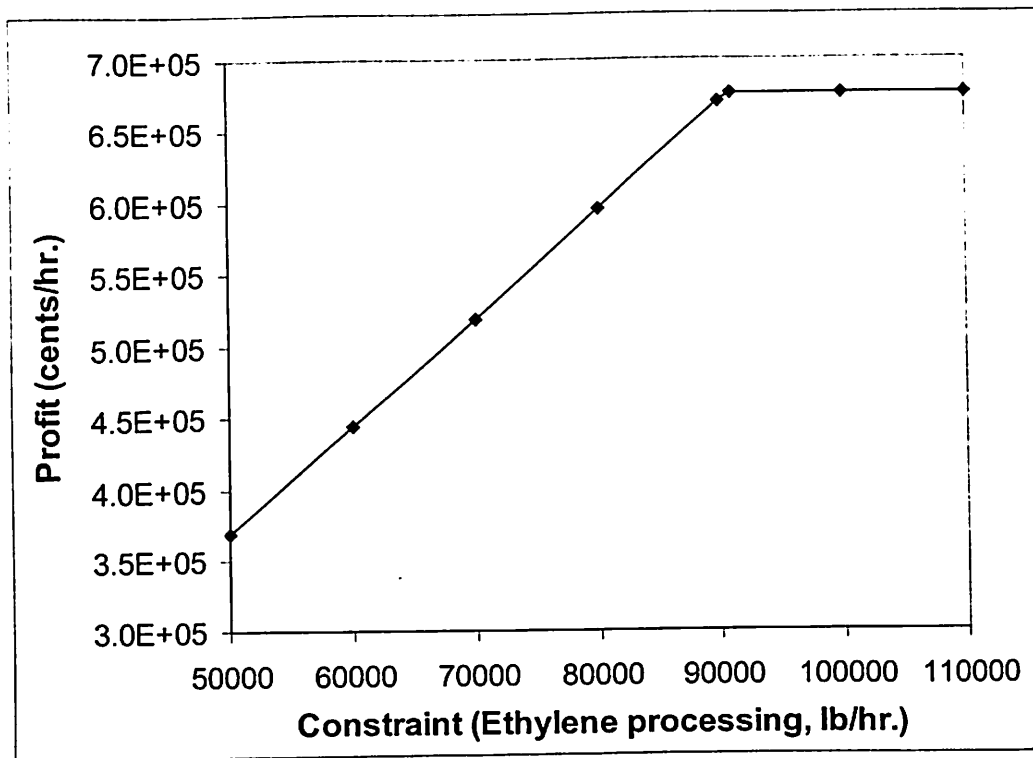


Fig. 3.13. Variation of Profit with ethylene processing constraint  
(1 lb = 0.454 kg; 1 cent/hr = 0.48 Rs/hr)

LP simplex solution has been crosschecked using a software package named *TORA*, [Taha, 1997], and the same results are obtained as shown in Table-3.7. Table 3.8 presents the comparison, in terms of the number of objective function evaluations, CPU-time and proportion of convergencies to the optimum, between the different DE strategies. In this table, *NFE*, *NRC* and CPU-time represents, respectively the mean number of objective function evaluations over all the 10 experiments, the percentage of convergencies to the global optimum and the average CPU time per experiments (key parameters used are:  $NP = 40$ ,  $CR = 0.9$ ,  $F = 0.6$ , accuracy = 0.0001%). From the Table-3.8, it is evident that the strategy number 7 is the best strategy. It takes least average CPU-time, maximum *NRC* and minimum *NFE*.

### 3.5.1.7.3. Results of DE & MDE

Results of reformulated problem using DE & MDE are shown in Table-3.9. In this table, *NFE*, *NRC* and CPU-time represents, respectively the mean number of

objective function evaluations over all the 100 experiments, the percentage of convergencies to the global optimum and the average CPU time per experiments (key parameters use are:  $NP = 40$ ,  $CR = 0.8$ ,  $F = 0.5$ , accuracy = 0.0001%).

**Table-3.9. Results of MDE and DE for Thermal cracker problem**

	DE <sup>#</sup>		MDE <sup>#</sup>	
	FBM	MWFB	FBM	MWFB
<i>NFE</i>	6978	21423	6496	18209
CPU-time (s)	0.2495	0.7681	0.2280	0.6632
<i>NRC</i>	100	100	100	100

<sup>#</sup>Strategy is DE/rand/1/bin

It is evident from above table that MDE out perform DE, i.e., faster than DE in locating global optimum. Also, it is to be noted that *NFE* or CPU-time is more in case of method *without forcing* the bound (in both DE & MDE) as compared to *forced bound* method. This is because that optimum value of several decision variables is lying on the boundary (i.e., on upper or lower limit of a decision variable).

### 3.5.1.8. Conclusions

In this section-3.5.1, the optimization of thermal cracker operation using DE, MDE and LP Simplex method has been presented. The strategy that took minimum CPU-time with highest *NRC* is strategy no. 7. The results obtained by all the three methods (MDE, DE & LP Simplex) are same and matches with that reported in literature. Both DE & MDE are found to be reliable in locating the global optimum with success rate (*NRC*) of 100%. MDE is found to be the better than DE in terms of function evaluations/ CPU-time.

## 3.5.2. Optimal Design Of An Auto-Thermal Ammonia Synthesis Reactor

### 3.5.2.1. Introduction

Ammonia is one of the most important chemicals produced as it enjoys the wide use in the manufacture of fertilizers. Hence modeling and simulation of ammonia manufacturing process has received considerable attention among the process

industries. Simulation models for ammonia synthesis converters of different types have been developed for design, optimization (Annable, 1952; Eymery, 1964; Dyson, 1965; Murase et al., 1970; Singh and Saraf, 1979; Upreti and Deb, 1997), and control (Shah, 1967) purposes.

Annable (1952) compared the performance of an auto-thermal ammonia synthesis reactor (Fig. 3.14) with the maximum yield that could be obtained if one had direct control of the temperature profile. He found that conversion could be increased by 14%. Obviously, one does not have direct control of the temperature profile, but it could be affected by the configuration of the heat transfer surface, *viz.*, added insulation and/or fins. Dyson (1965) considered the general problem of determining the heat transfer coefficient *vs.* length function that would maximize the yield in an autothermal reactor.

Murase et al. (1970) computed the optimum temperature trajectory along the reactor length applying the Pontryagin's maximum principle. Edgar and Himmelblau (1989) used Lasdon's generalized reduced-gradient method to arrive at an optimal reactor length corresponding to a particular reactor top temperature of 694 K. However they also ignored a term mentioned in Murase's formulation, pertaining to the cost of ammonia already present in the feed gas, in the objective function. Also the expressions of the partial pressures of nitrogen, hydrogen and ammonia, used to simulate the temperature and flow rate profiles across the length of the reactor, were not correct.

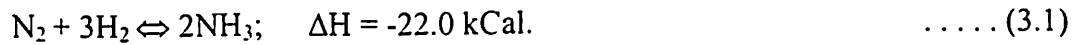
Upreti and Deb (1997) used Murase's formulation with correct objective function and correct stoichiometric expressions of the partial pressures of  $N_2$ ,  $H_2$ , and  $NH_3$ . They used simple GA in combination with Gear package of NAG library's subroutine, D02EBF, for the optimization of ammonia synthesis reactor. They

obtained mass flow rate of nitrogen, feed gas temperature and reaction gas temperature at every 0.01 m of 10 m reactor length. Also, there is a contradiction in the temperatures & gas flow rate profiles obtained. They reported the profiles that were not smooth as in earlier literature. Moreover, they reported reverse reaction condition at the top temperature of 664 K, which was not found in literature earlier. Hence, the present study (Babu and Angira, 2005a; Angira and Babu, 2005d) is carried out in order to take care of the above deficiencies.

The three-coupled differential equations are solved using three different techniques (to check if it was the limitation of numerical methods because of which reverse reaction trend was not reported earlier), namely, Euler's method, Runge-Kutta method (fourth order) with fixed step size (Kreyszig, 1993), and NAG subroutine (D02EJF) in MATLAB (version 5.1). The small step sizes of 0.01 and 0.001 are used for above said numerical methods while Upreti and Deb (1997) used the step size of 0.01 only. In addition, a software package POLYMATH (Himmelblau, 1997) was also used for the same. Finally, Quasi-Newton method, Secant method and Genetic Algorithm (GA) and Differential Evolution (DE) were used for optimization in combination with above three numerical methods, and the results are compared.

#### **3.5.2.2. *Auto-Thermal Ammonia Synthesis Reactor***

A typical auto-thermal ammonia synthesis reactor is shown in Fig. 3.14. Typically, the ammonia manufacturing process consists of production of synthesis gas from the petroleum feed stock, compression of the gas to the required pressure, and the synthesis loop in which the conversion to ammonia takes place. The ammonia converter is part of the synthesis loop, and its operation is quite crucial in the overall control strategy of the plant. In the converter, the following catalytic reaction takes place at elevated temperatures and pressures releasing a large amount of heat.



This heat has to be removed to obtain a reasonable conversion as well as to protect the catalyst life. At the same time, the released heat energy is utilized to heat the incoming feed-gas to proper reaction temperature.

The reaction zone (shaded portion of Fig. 3.14) contains the catalyst. A number of cooling tubes are inserted vertically through the reaction zone. The feed gas comes in from the lower part of the reactor and flows up through the top of the reactor. Then, changing direction, it flows down through the reaction zone and heat exchanger to the outlet. As with all reversible exothermic reactions, the temperature, at which the reaction rate is maximum, decreases as the conversion increases. Even though one would like to maximize the reaction rate at each instant, it is impossible to obtain the ideal temperature profile by control of available design variables. The countercurrent flow does, however, cause the temperature to decrease in the bottom part of the reactor because of heat transfer between the reacting gas and feed gas.

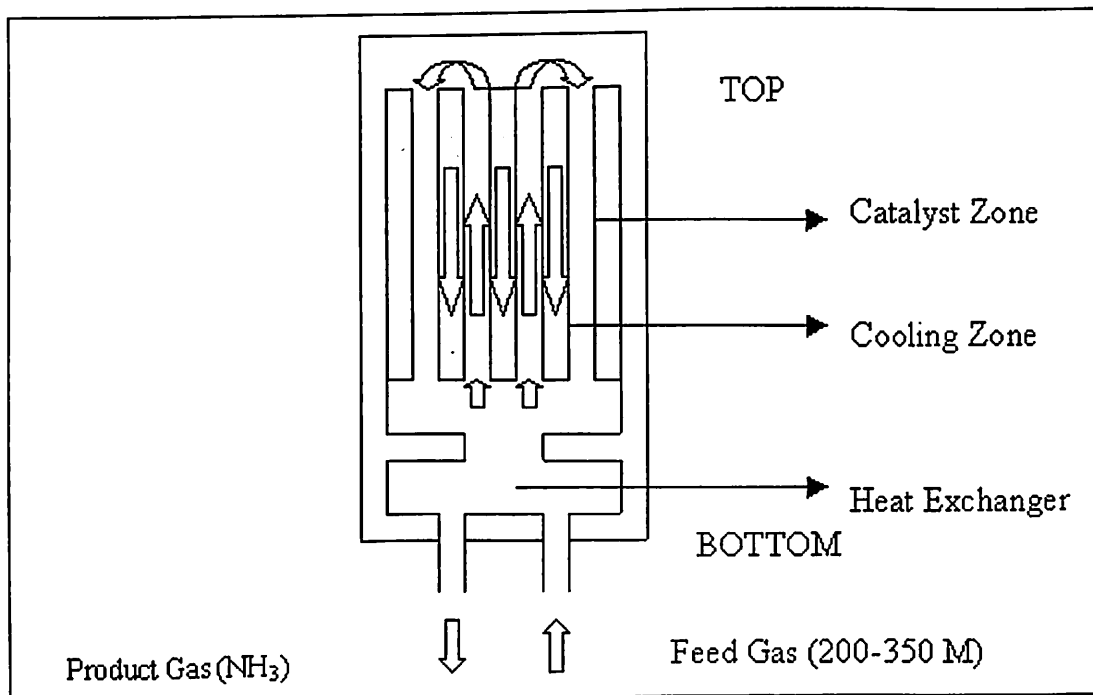


Fig. 3.14. Auto-thermal ammonia synthesis reactor



### 3.5.2.3. *Problem Formulation*

The Problem formulation is similar to that given in Murase et al. (1970) including the modifications mentioned in Upreti and Deb (1997). Upreti and Deb (1997) in their paper cited Edgar and Himmelblau (1989) for model equations, but the former did not give them. Unfortunately, equations (c) and (d) on page 545 of the latter are incorrect where “1.5” is given as the coefficient. In fact, it should be in the power of  $p_{H_2}$ , as stated in equation (5) of this paper. The same error was found in Murase *et al.* (1970) also. Hence, for the sake of clarity, it becomes important to give the correct model equations. The correct model equations are presented in section-3.5.2.3.2.

Feed gas contains 21.75 mole% nitrogen, 65.25 mole% hydrogen, 5 mole% ammonia, 4 mole% methane and 4 mole% argon. In a typical ammonia synthesis reactor (Fig. 3.14), feed gas enters the reactor from the bottom. The yield of ammonia depends on the temperature of feed gas at the top of the reactor (henceforth called top temperature), the partial pressures of the reactants (nitrogen and hydrogen), and the reactor length. The following assumptions are made in modeling the auto-thermal ammonia synthesis reactor (Murase et al., 1970):

1. The rate expression is valid.
2. The model is one-dimensional, i.e., the temperature and concentration gradients in the radial direction are neglected.
3. Heat & mass diffusion in the longitudinal direction are negligible.
4. The temperature of the gas flowing through the catalyst zone is equal to the reacting gas and the catalyst particles.
5. The heat capacities of the reacting gas and the feed gas are constant.
6. The catalyst activity is uniform along the reactor and equal to unity.

7. Pressure drop across the reactor is negligible compared to the total pressure of the system.

### 3.5.2.3.1. Objective function

The objective function is the economic return based on the difference between the value of the product gas (heating value and the ammonia value) and the value of feed gas (as a source of heat only) less the amortization of reactor capital costs. Other operating costs are omitted, as their contribution is not significant. As shown in Upreti and Deb (1997), the final corrected objective function  $F = f(x, N_{N_2}, T_f, T_g)$  is:

$$F = 1.33563 \times 10^7 - 1.70843 \times 10^4 N_{N_2} + 704.09(T_x - T_0) - 699.27(T_f - T_0) - [3.45663 \times 10^7 + 1.98865 \times 10^9 x]^{0.5} \dots (3.2)$$

It is clear from the above expression that the objective function depends on four variables: the reactor length  $x$ , proportion of nitrogen  $N_{N_2}$ , the reacting gas temperature  $T_g$ , and the feed gas temperature  $T_f$ , for a given top temperature  $T_0$ .

### 3.5.2.3.2. Constraints

As stated earlier, there are mistakes in modeling equations reported by Murase *et al.* (1970), some of which (partial pressure equations) were corrected and reported by Upreti and Deb (1997). However, the mistakes in material balance equation were not reported. Hence for the sake of clarity, all the corrected modeling equations are presented below:

#### (a) Energy-Balances

**Feed Gas:** Referring to Fig. 3.15, an energy balance on the feed stream yields the following equation:

$$\frac{dT_f}{dx} = \frac{US_1}{WC_{pf}} (T_g - T_f) \dots (3.3)$$

Where

$U$  = Overall heat transfer coefficient, Kcal/m<sup>2</sup>.hr.K

$S_1$  = Surface area of cooling tubes per unit length of reactor, m

$T_g$  = Temperature of reacting gas, K

$W$  = Total mass flow rate, kg/hr

$C_{pf}$  = Specific heat of feed gas, kcal/kg. K

$x$  = Distance along axis, m

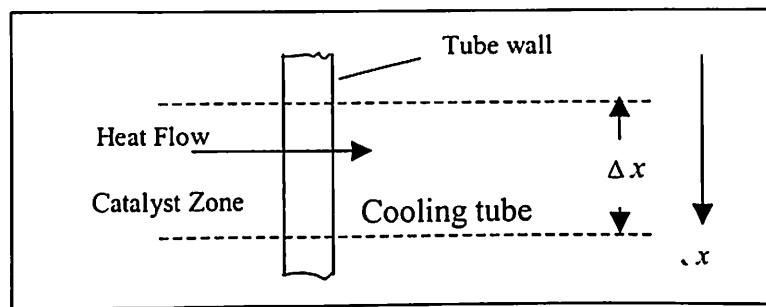


Fig. 3.15. Energy and material balance on control volume

**Reacting Gas:** Similarly, for the reacting gas,

$$\frac{dT_g}{dx} = -\frac{US_1}{WC_{pg}}(T_g - T_f) + \frac{(-\Delta H)S_2}{WC_{pg}}\left(\frac{-dN_{N_2}}{dx}\right) \quad \dots\dots (3.4)$$

Where

$\Delta H$  = Heat of reaction, kcal/kg mole of N<sub>2</sub>

$S_2$  = Cross-sectional area of catalyst zone, m<sup>2</sup>

$\frac{dN_{N_2}}{dx}$  = Reaction rate, kg moles of N<sub>2</sub> /hr.m<sup>3</sup>

$C_{pg}$  = Heat capacity of reacting gas, kcal/kg. K

(b) *Material balance*

Considering the incremental distance in the catalyst zone (Fig. 3.15) and performing a N<sub>2</sub> material balance yields:

$$\frac{dN_{N_2}}{dx} = -f \left[ k_1 \frac{P_{N_2} P_{H_2}^{1.5}}{P_{NH_3}} - k_2 \frac{P_{NH_3}}{P_{H_2}^{1.5}} \right] \quad \dots (3.5)$$

Where,

$f$  = Catalyst activity

$P_{N_2}, P_{H_2}, P_{NH_3}$  = Partial pressure of  $N_2, H_2,$  and  $NH_3$

$k_1, k_2$  = Rate constants;

$$k_1 = 1.78954 \times 10^4 \exp\left(\frac{-20800}{RT_g}\right) \quad (3.6)$$

$$k_2 = 2.5714 \times 10^{16} \exp\left(\frac{-47400}{RT_g}\right) \quad (3.7)$$

In order to maintain the energy & material balance of reaction, the above three coupled differential equations (Eqs. 3.3, 3.4, and 3.5) must be satisfied. It turns out that three of the above variables ( $T_f, T_g, N_{N_2}$ ) can be eliminated by satisfying these energy & material balance equations. Thus, practically, there is only one design variable for given top temperature. The partial pressures appearing in the differential equations are computed as follows:

$$P_{N_2} = \frac{286 N_{N_2}}{2.598 N_{N_2}^0 + 2 N_{N_2}} \quad (3.8)$$

$$P_{H_2} = 3 P_{N_2} \quad (3.9)$$

$$P_{NH_3} = \frac{286(2.23 N_{N_2}^0 - 2 N_{N_2})}{2.598 N_{N_2}^0 + 2 N_{N_2}} \quad (3.10)$$

The boundary conditions are:

$$T_f = T_0 \text{ at } x=0 \quad (3.11)$$

$$T_g = T_f \text{ at } x=0 \quad (3.12)$$

$$N_{N_2}^0 = 701.2 \text{ kmol/hr.m}^2 \text{ at } x=0 \quad (3.13)$$

The three inequality constraints that limit the values of three of the design variables are as given below:

$$0.0 \leq N_{N_2} \leq 3220 \quad (3.14)$$

$$400 \leq T_f \leq 800 \quad (3.15)$$

$$0.0 \leq x \leq 10.0 \quad (3.16)$$

Since the reaction gas temperature ( $T_g$ ) depends on the nitrogen mass flow rate ( $N_{N_2}$ ), feed gas temperature ( $T_f$ ) and reactor length ( $x$ ), explicit bounds on  $T_g$  are not necessary. From the system model, we have three differential equations and four variables, making the degrees of freedom equal to one. We specify the length of the reactor, calculate the remaining variables using the system model and then pass these variables to the optimization routine. The computation procedure for the optimization carried out is shown in Fig. 3.16.

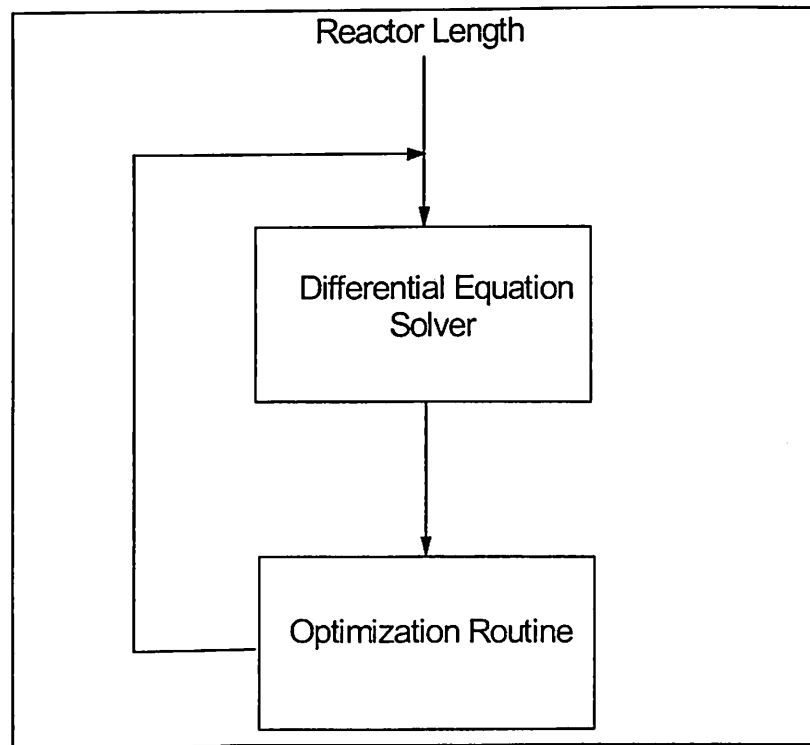


Fig. 3.16. Computation procedure

### 3.5.2.4. Results & Discussion

#### 3.5.2.4.1. Temperature & Flow rate Profiles

First, the system equations (3.3, 3.4, and 3.5) were solved using Runge-Kutta fixed step size method (RKFS) and Fig. 3.17 shows the profiles obtained. From the graph it is clear that the profiles of  $N_{N_2}$  &  $T_f$  intersect at a reactor length of 4.935 m. And the profiles of  $N_{N_2}$  &  $T_g$  intersect at a reactor length of 8.913 m. It is evident from the graph that the profiles are smooth and there are no spikes as reported by Upreti and Deb (1997). Therefore, in order to check if it is the limitation of RKFS because of which spikes are not obtained, the other numerical methods viz. NAG subroutine (D02EJF) in MATLAB and Euler's method (EULER), are also used for simulating the results. Apart from above mentioned numerical methods, the POLYMATH (a software package) is also used to simulate the results for profiles.

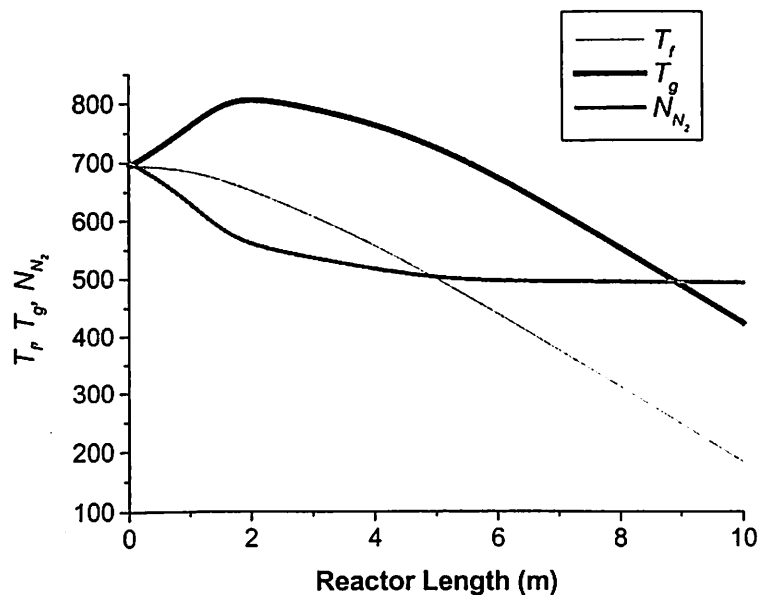


Fig. 3.17. Profile obtained using RKFS (step size = 0.01)

Figs. 3.18, 3.19, and 3.20 show the profiles obtained using the methods mentioned above. Fig. 3.21 shows the profiles obtained using D02EJF with small step size of 0.001 to ensure that we may not miss spikes if any. The profiles obtained are quite

smooth, without any spikes and are similar to those obtained with RKFS. The profiles are same qualitatively but they differ slightly quantitatively. To illustrate the exact difference quantitatively, the above observations & comparison of various numerical methods used for simulating the results are presented in Table-3.5.8 for the reactor lengths of 10 m. Table-3.5.9 & Table-3.5.10 show the reactor length for which the variables  $N_{N_2}$  &  $T_g$  and  $N_{N_2}$  &  $T_f$  intersect respectively.

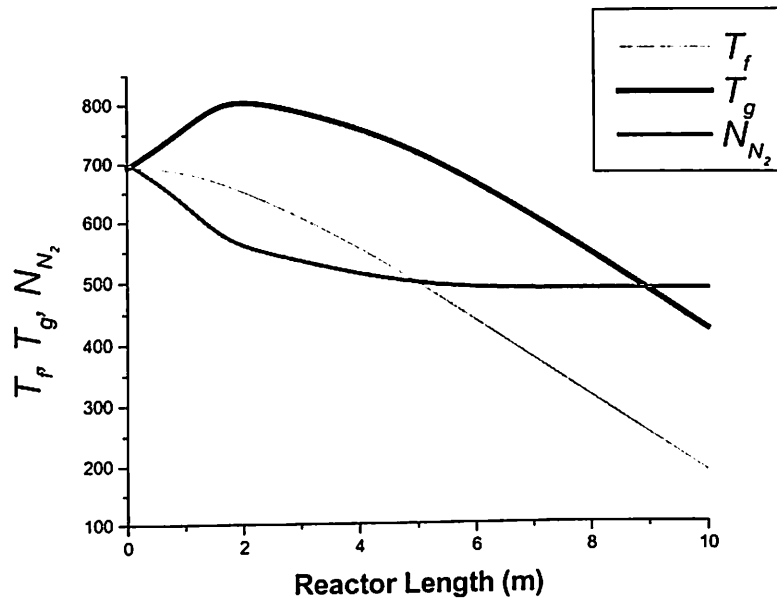


Fig. 3.18. Profile obtained using EULER (step size = 0.01)

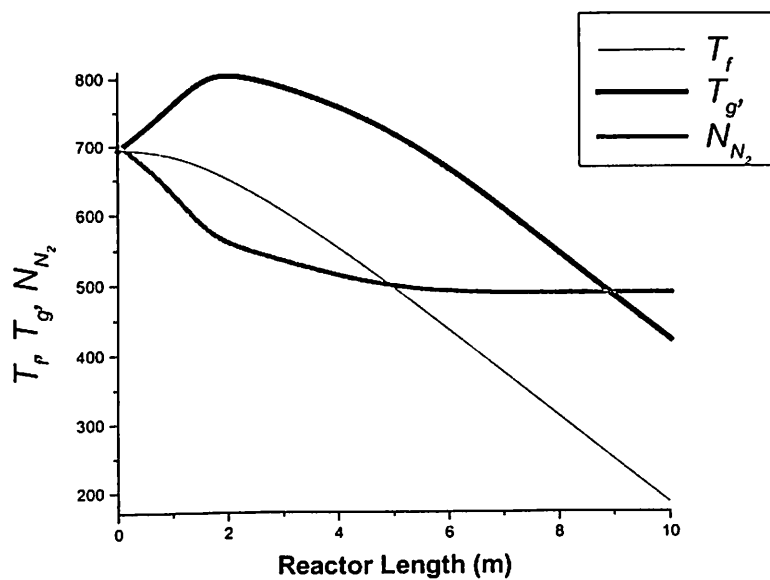


Fig. 3.19. Profile obtained using POLYMATH (step size = 0.01)

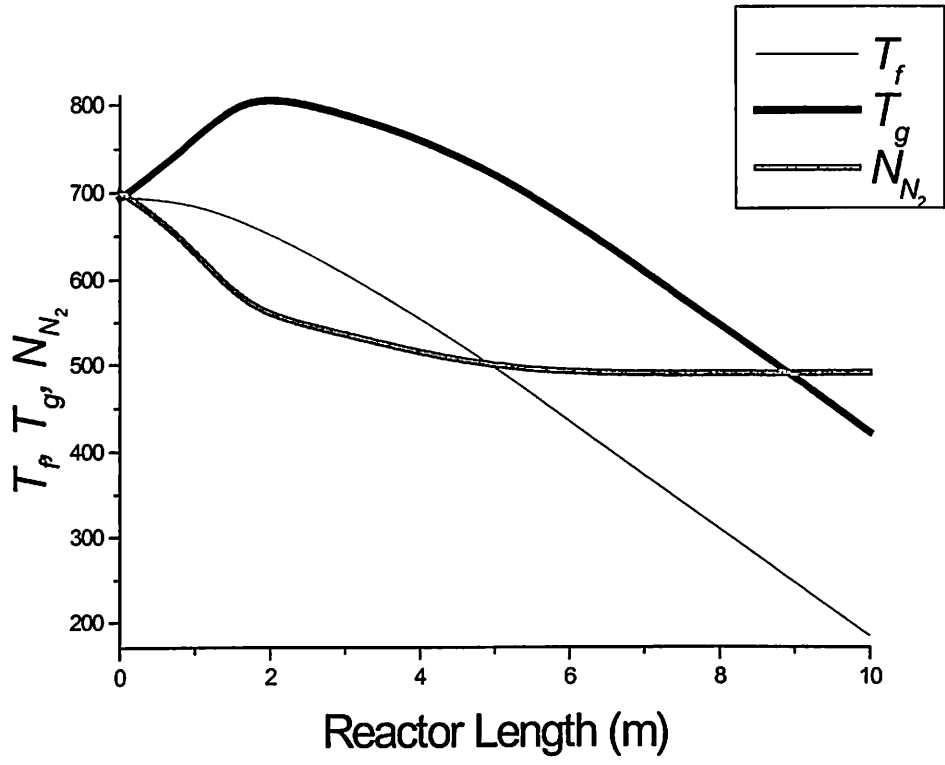


Fig. 3.20. Profile obtained using D02EJF (step size = 0.01)

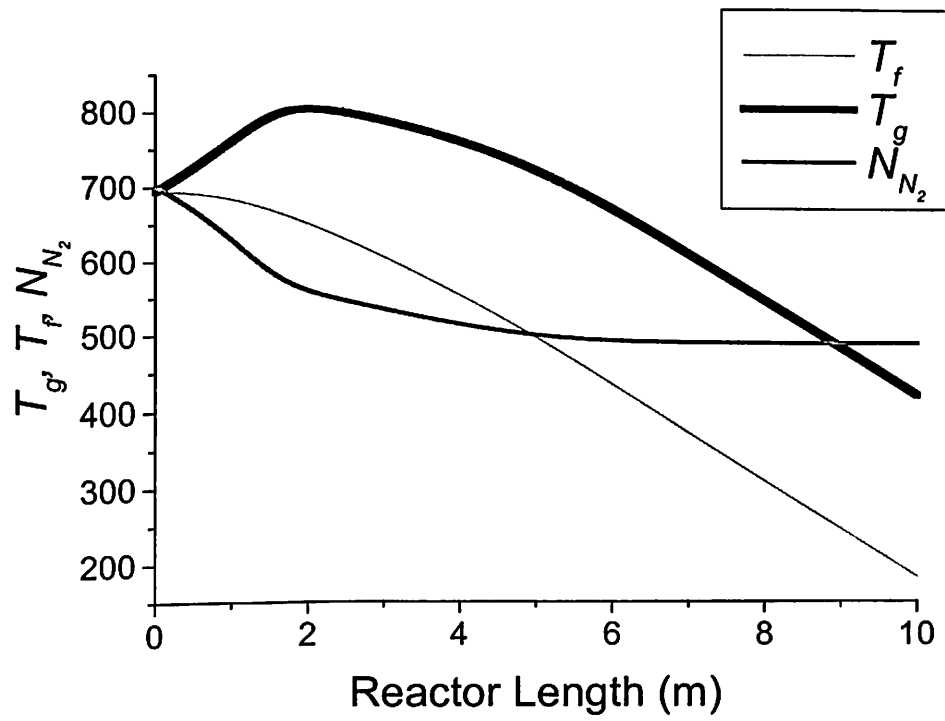


Fig. 3.21. Profile obtained using D02EJF (step size = 0.001)



**Table-3.10. Comparison of different numerical methods at  $x = 10$  m**

Methods Parameters	RKFS <sup>4</sup>	EULER <sup>4</sup>	POLYMATH <sup>4</sup>	D02EJF <sup>4</sup>
$x$	10.00	10.00	10.00	10.00
$N_{N_2}$	490.46	490.16	490.46	490.46
$T_g$	422.25	424.33	422.25	422.25
$T_f$	183.37	187.96	183.38	183.39

<sup>4</sup>Step size = 0.01

**Table-3.11. Reactor length at which variables  $N_{N_2}$  &  $T_g$  intersect**

Methods Parameters	RKFS <sup>5</sup>	EULER <sup>5</sup>	POLYMATH <sup>5</sup>	D02EJF <sup>5</sup>
$x$	8.913	8.916	8.913	8.913
$N_{N_2}, T_g$	490.46	490.44	490.46	490.46

<sup>5</sup>Step size used is 0.001.

**Table-3.12. Reactor length at which variables  $N_{N_2}$  &  $T_f$  intersect**

Methods Parameters	RKFS <sup>6</sup>	EULER <sup>6</sup>	POLYMATH <sup>6</sup>	D02EJF <sup>6</sup>
$x$	4.935	4.939	4.934	4.934
$N_{N_2}, T_f$	501.47	501.39	501.49	501.49

<sup>6</sup>Step size used is 0.001.

From Table-3.10, Table-3.11 and Table-3.12, it is evident that all the numerical methods (stated above) are equally good barring a few of the following differences. There is good agreement between RKFS & POLYMATH, which can be explained with the fact that POLYMATH software is based on RKFS algorithm. D02EJF also gave the same results as those obtained by using RKFS & POLYMATH. EULER is giving almost same results for step size of 0.001, but for step size of 0.01 the results are slightly different from RKFS, POLYMATH & D02EJF. Even for step size of 0.01, the difference in the prediction between EULER & RKFS/POLYMATH/D02EJF respectively, is insignificant at reactor length of 10m.

Similarly, the difference in prediction of intersections (Table-3.11 and 3.12) is also negligible between EULER & RKFS/POLYMATH/D02EJF method. Based on the above observations, it can be substantiated that all the three methods are equally good both qualitatively giving the same results and quantitatively with slight difference (for step size of 0.01). Hence, any one of the above numerical methods can be used for the solution of three coupled differential equations.

Upreti and Deb (1997) reported that the reverse reaction predominated the forward reaction at the top temperature of 664 K. In the present study, all the six methods quoted above were used to generate temperature & flow rate profiles at that temperature. Surprisingly, there is no such trend observed in the profiles obtained [for which Upreti and Deb (1997) gave a very good physical explanation] as can be seen in Fig. 3.22 plotted using the results obtained with RKFS.

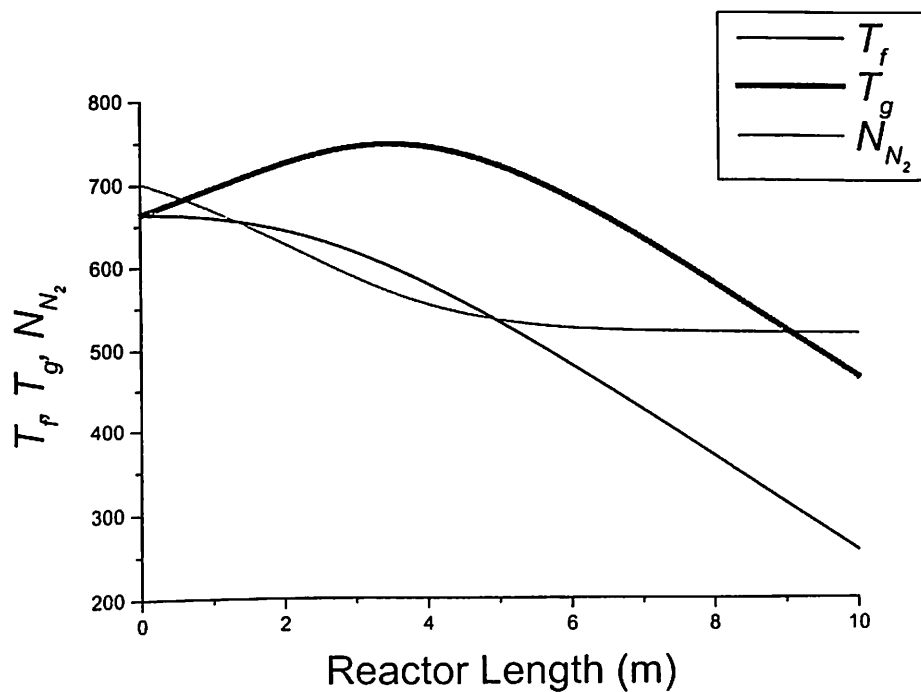


Fig. 3.22. Profile obtained using RKFS for top temperature of 664 K

To see the presence of any reverse reaction effect, the program is executed for temperatures even below 664 K with interval of 10 K up to 600 K. But no such trend

is observed. Typical results obtained at top temperature of 640 K and 600 K are shown in Figs. 3.23 & 3.24 respectively. As is evident from the figures there is no reverse reaction effect even at a top temperature as low as 600 K.

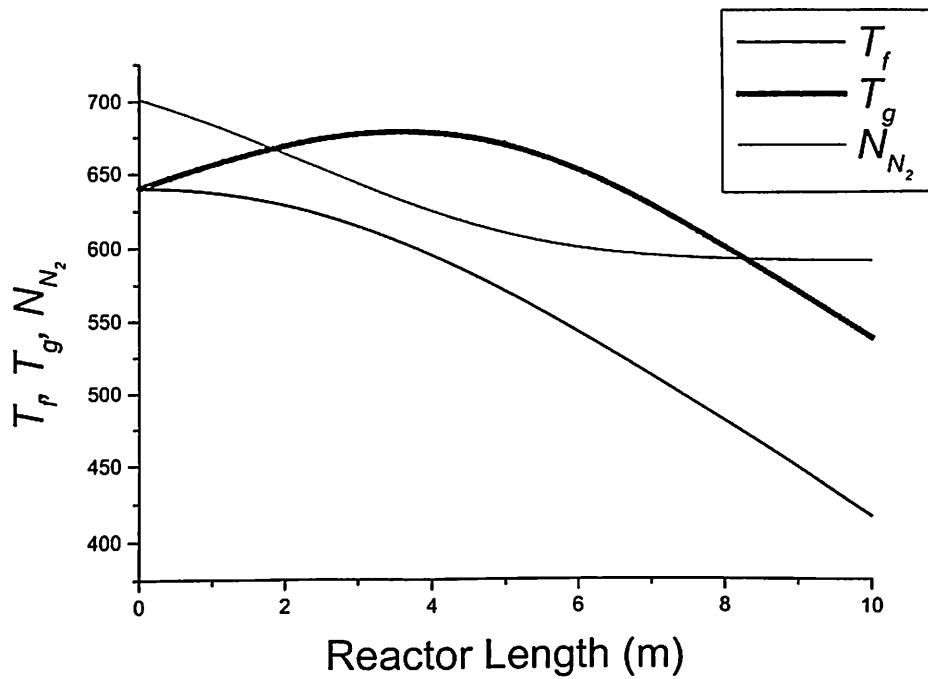


Fig. 3.23. Profile obtained using RKFS for top temperature of 640 K

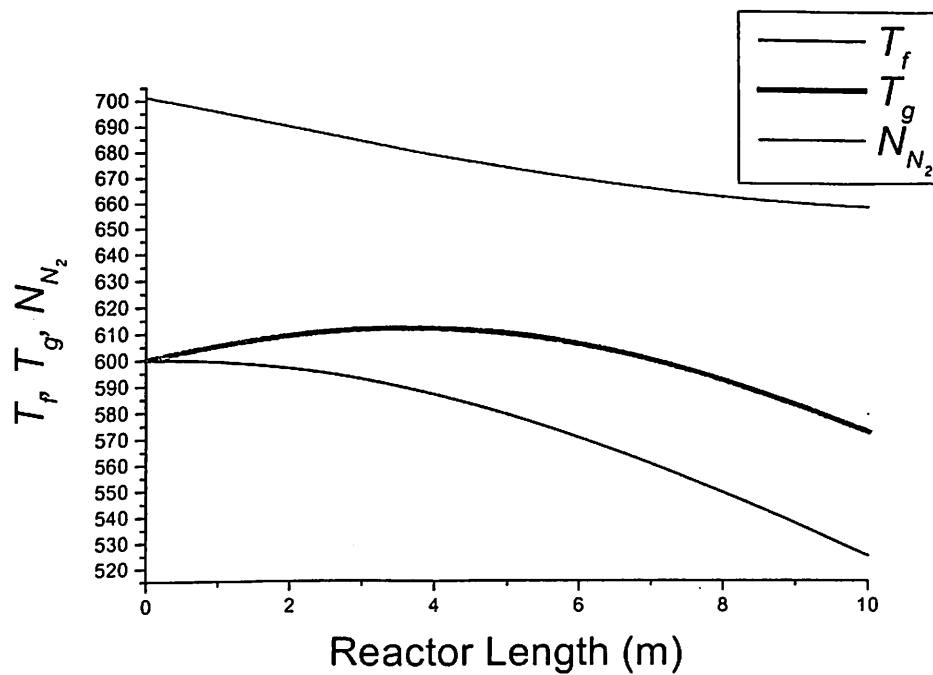


Fig. 3.24. Profile obtained using RKFS for top temperature of 600 K

Also, Upreti and Deb (1997) reported that the three differential equations (3.3, 3.4, and 3.5) become unstable at the top temperature of 706 K. However, using the above stated numerical methods, it is found that the equations are not unstable even at a top temperature as high as 800 K. It may be noted that Upreti and Deb (1997) used NAG library's subroutine D02EBF (which is now replaced by D02EJF (NAG, 2004)).

The possibility of using wrong equation by Upreti and Deb (1997) is explored (i.e. taking '1.5' as coefficient, as referred in the first paragraph of section-3.5.2.3 in this chapter) and neither spikes nor reverse reaction effect were found even with this. So it may be because of the problem in the software package that they reported the reversible reaction effect. The numerical methods used in the present study include the replaced NAG subroutine (D02EJF) as well, which did not give the above reverse reaction effect (Fig. 3.25).

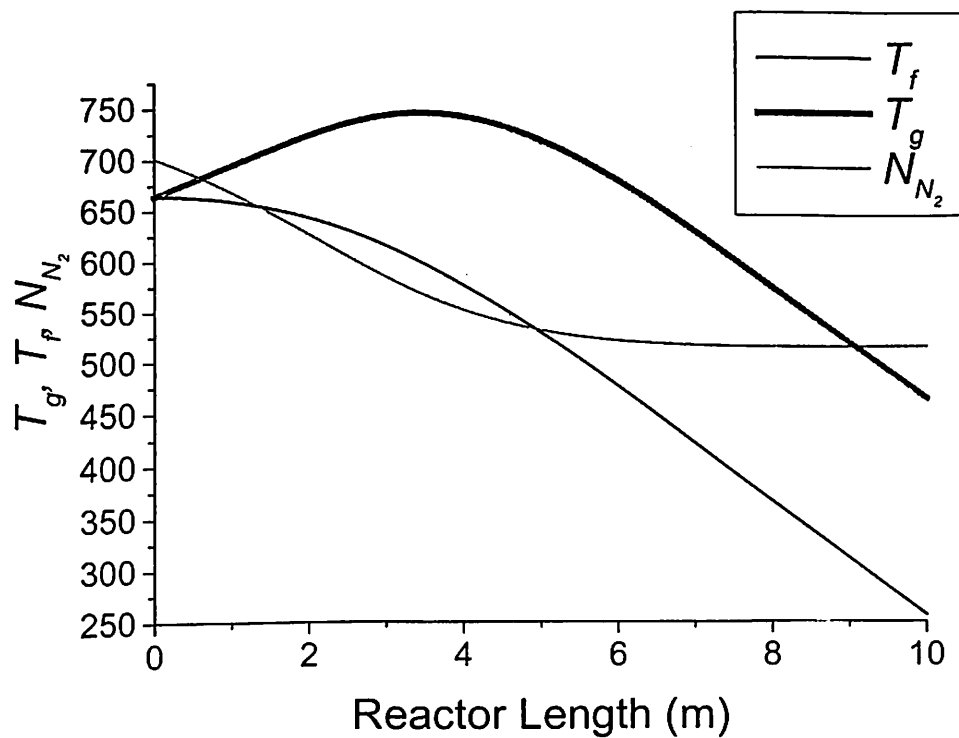


Fig. 3.25. Profile obtained using D02EJF for top temperature of 664

#### 3.5.2.4.2. Optimization of Reactor Length

Fig. 3.26 shows the variation of objective function with reactor length (without any constraint on  $T_f$ ). The value of objective function increases with reactor length. From the Fig. 3.26, it is clear that POLYMATH, RKFS & D02EJF show same variation in objective function with reactor length though EULER (step size = 0.01) shows slightly different variation (Fig. 3.27). Though to a naked eye, the profit function (objective function) appears to be monotonously increasing, magnification of highlighted part clearly indicates that there exists a single maximum as shown in Fig. 3.27.

The effect of top temperature on objective function for various reactor lengths is shown in Fig. 3.28. Higher the top temperature, higher is the objective function (profit). Also, after a reactor length of 7.0 m, the objective function almost becomes constant for a given top temperature, as the decrease in its value with length is insignificant. The maximum difference in objective function value for top temperature of 706 & 666 K occurs at a reactor length of 1.47 m. This is about 36.42%. At a reactor length of 7 m & 10 m, this difference is about 7.5% & 7.35% respectively which is almost constant.

Fig. 3.29 shows the optimum reactor length at various top temperatures using EULER, RKFS & D02EJF methods. The optimum reactor length varies with top temperature while Upreti and Deb (1997) reported it to be almost constant. Murase *et al.* (1970) also obtained qualitatively same variations but quantitatively different (which could be due to incorrect partial pressure equations they used) as shown in Fig. 3.29.

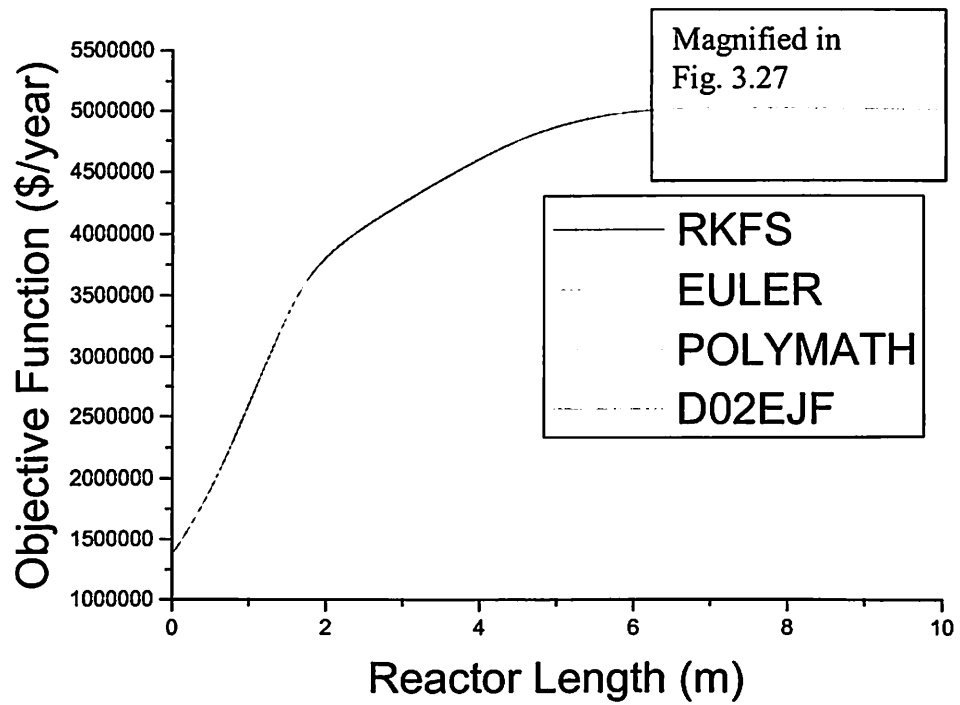


Fig. 3.26. Variation of objective function with reactor length (various numerical methods)

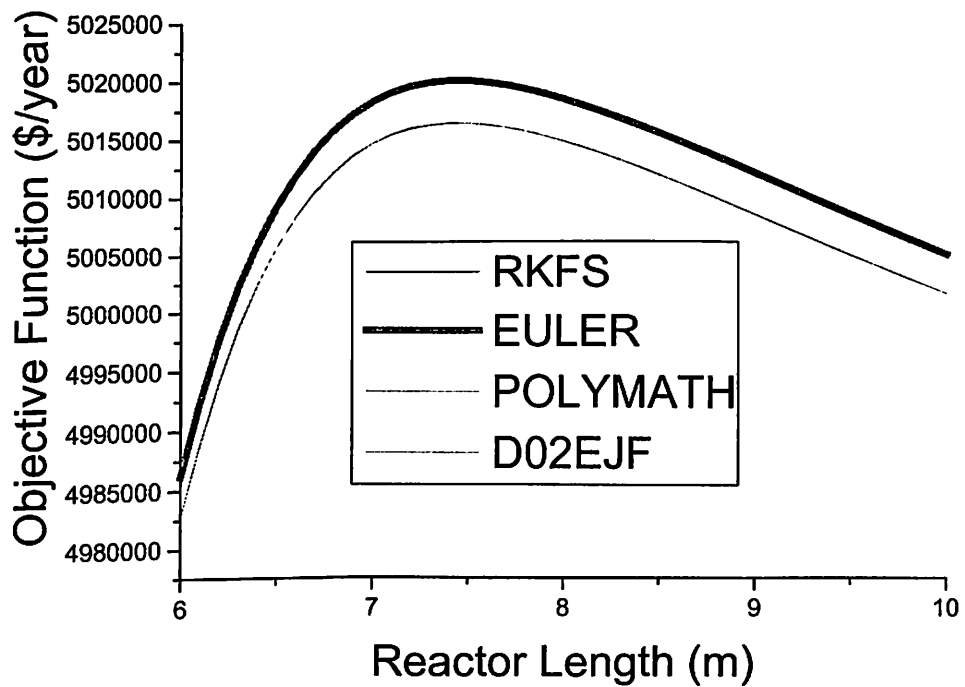


Fig. 3.27. Magnification of highlighted part of Fig. 3.26

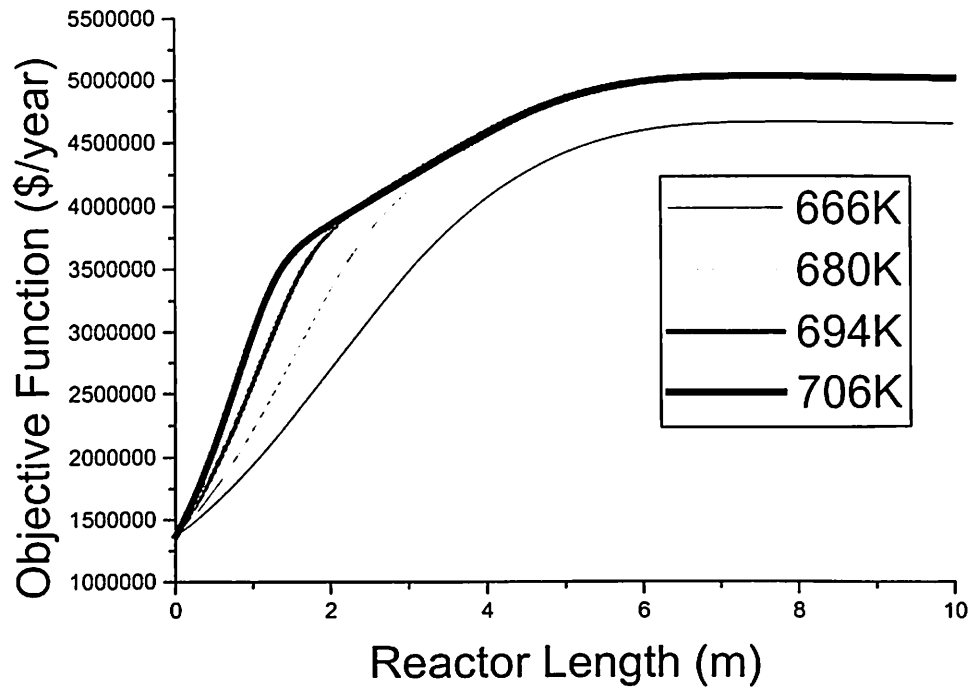


Fig. 3.28. Objective function variation with reactor length at various top temperatures using RKFS

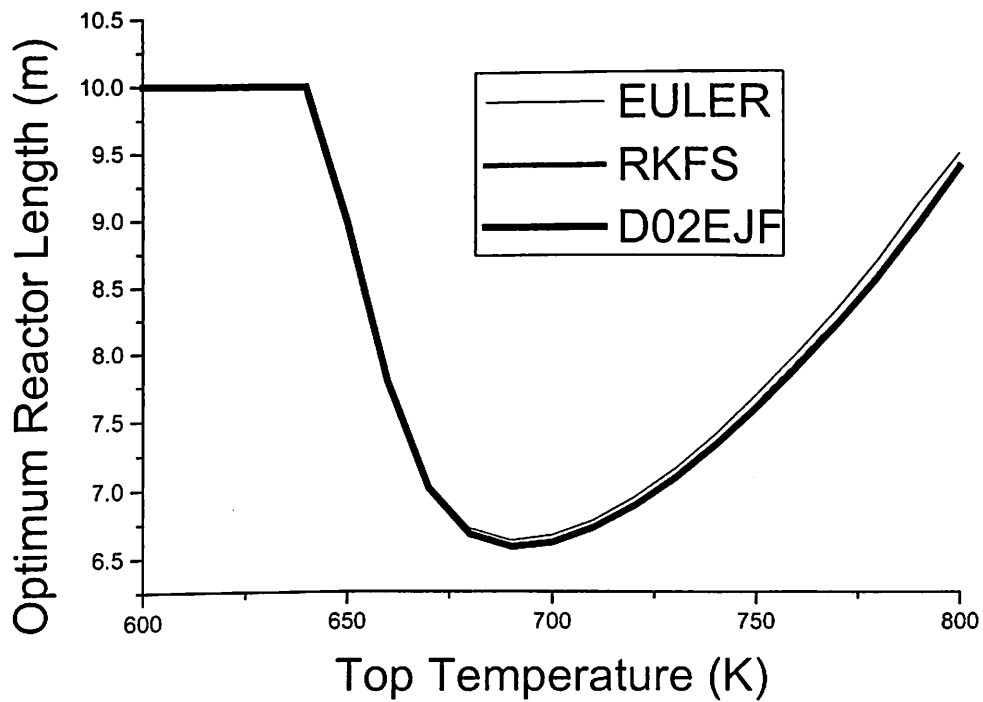


Fig. 3.29. The variation of optimum reactor length with top temperature

It is evident from Figs. 3.26 and 3.27 that the objective function profile is smooth and unimodal. Therefore, any simple gradient-based method can be used for optimization as against the population based stochastic optimization technique such as GA being used by Upreti and Deb (1997). In the present paper Quasi-Newton (QN) and Secant methods are used. GA, DE and MDE are also applied for academic interest only.

Table-3.13 and Table-3.14 show the results obtained from different methods & the comparison with those obtained by Murase *et al.* (1970), Edgar and Himmelblau (1989) & Upreti and Deb (1997) using Pontryagin's maximum principle (PMP), Lasdon's generalized reduced-gradient method (LGRG) & Genetic Algorithm (GA) respectively.

**Table-3.13. Optimal Length and Profit obtained from various methods**

Method	SECANT		GA		DE		
	RKFS*	EULER*	RKFS*	EULER*	RKFS*	EULER*	D02EJF*
Optimal Length (m)	6.587	6.591	6.587	6.591	6.586	6.591	6.586
Profit (\$/yr) x 10 <sup>6</sup>	5.0	5.0	5.0	5.0	5.0	5.0	5.0

\* Step size = 0.001

**Table-3.14. Comparison with literature values**

Methods Parameters ↓	PMP (Murase et al, 1970)	LGRG (Edgar and Himme- blau, 1989)	D02EBF With GA (Upreti and Deb, 1997)	EULER* with QN	RKFS* with QN	D02EJF* with QN	D02EJF* with DE	RKFS* with DE	RKFS* with MDE
Optimum <i>x</i> (m)	5.18	2.58	5.33	6.63	6.59	6.59	6.586	6.59	6.59
Objective function (million \$/year) x10 <sup>6</sup>	Not Reported	1.29	4.23	5.0	5.0	5.0	5.0	5.0	5.0

\* Step size = 0.01

From Table-3.13 and Table-3.14, it is clear that irrespective of the optimization technique (viz. GA, DE, MDE, SECANT, and QN) the optimum reactor length is



same, i.e., 6.59m. Also, by decreasing the step size from 0.01 to 0.001, the accuracy does not vary but computational time increases. So a step size of 0.01 is sufficient to get accurate optimum reactor length. From Table-3.14, we observe that an optimum reactor length of 2.58 m is reported by Edgar and Himmelblau (1989) and 5.18 m by Murase *et al.* (1970), both of which are wrong due to the errors in their problem formulations as pointed out by Upreti and Deb (1997). An optimum reactor length of 5.33 m and the corresponding objective function value is  $4.23 \times 10^6$  \$/year, reported by Upreti and Deb (1997) are also not correct as found in the present study. Among other methods, EULER method with a step size of 0.01 gives the same objective function value as that of RKFS though the optimum reactor length is slightly different. But with step size of 0.001 (see Table-3.13), the optimum reactor length obtained using EULER method is almost the same as obtained by RKFS & D02EJF methods. RKFS is a widely accepted method for solution of ordinary differential equations. In addition, backward differentiation formula (BDF) method (implemented in the NAG subroutine D02EJF) is a trusted method for solution of ODE. Hence, the correct optimum reactor length can be considered as 6.59 m with an objective function value of  $5.00 \times 10^6$  \$/year.

#### 3.5.2.5. *Conclusions*

In this section, the problem of optimal design of an ammonia synthesis reactor has been solved. Two numerical methods and NAG subroutine D02EJF in MATLAB for solving model equations, and QN, Secant, GA, DE & MDE for optimization, are used. POLYMATH is also used for solving three ODEs. Results indicate that the profiles of temperatures & flow rate are smooth and there is no reverse reaction effect irrespective of numerical method used for the solution of differential equations. A possible error in the integrator of the NAG subroutine used in the past study is

identified. And the new NAG subroutine shows a unimodal characteristic of the objective function, compared to a multimodal one found in the past study. The optimum reactor length depends upon the top temperature. At the top temperature of 694 K, the reactor length of 6.59 m was found to give the optimum objective function value of \$ 5.0x10<sup>6</sup>/year irrespective of the numerical methods and optimization techniques used.

### 3.5.3. Reactor Network Design (RND) Problem

This problem arises from the area of chemical engineering, and represent difficult non-linear optimization problem, with equality & inequality constraints. Comparison is made with  $\alpha$ BB algorithm (Adjiman et al., 1998a; 1998b), which can be used to solve problems belonging to the broad class of twice-differentiable constrained NLPs. The  $\alpha$ BB algorithm is based on a branch-and-bound approach, where a lower bound on the optimal solution is obtained at each node through the automatic generation of a valid convex underestimating problem.

#### 3.5.3.1. The Problem

This example, taken from Ryo & Sahinidis (1995), is a reactor network design problem, describing the system shown in Fig. 3.30. It involves the design of a sequence of two CSTRs where the consecutive reaction  $A \rightarrow B \rightarrow C$  takes place. The goal is to maximize the concentration of product  $B$  ( $x_4 = C_{B2}$ ) in the exit stream. This problem is known to have caused difficulties for other global optimization methods.

$$\text{Min } -x_4$$

Subject to

$$x_1 + k_1 x_1 x_5 = 1$$

$$x_2 - x_1 + k_2 x_2 x_6 = 0$$

$$x_3 + x_1 + k_3 x_3 x_5 = 1$$

$$x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 = 0$$

$$x_5^{0.5} + x_6^{0.5} \leq 4$$

$$(0, 0, 0, 0, 10^{-5}, 10^{-5}) \leq (x_1, x_2, x_3, x_4, x_5, x_6) \leq (1, 1, 1, 1, 16, 16).$$

Where  $k_1 = 0.09755988$

$$k_2 = 0.99k_1$$

$$k_3 = 0.0391908$$

$$k_4 = 0.9k_3$$

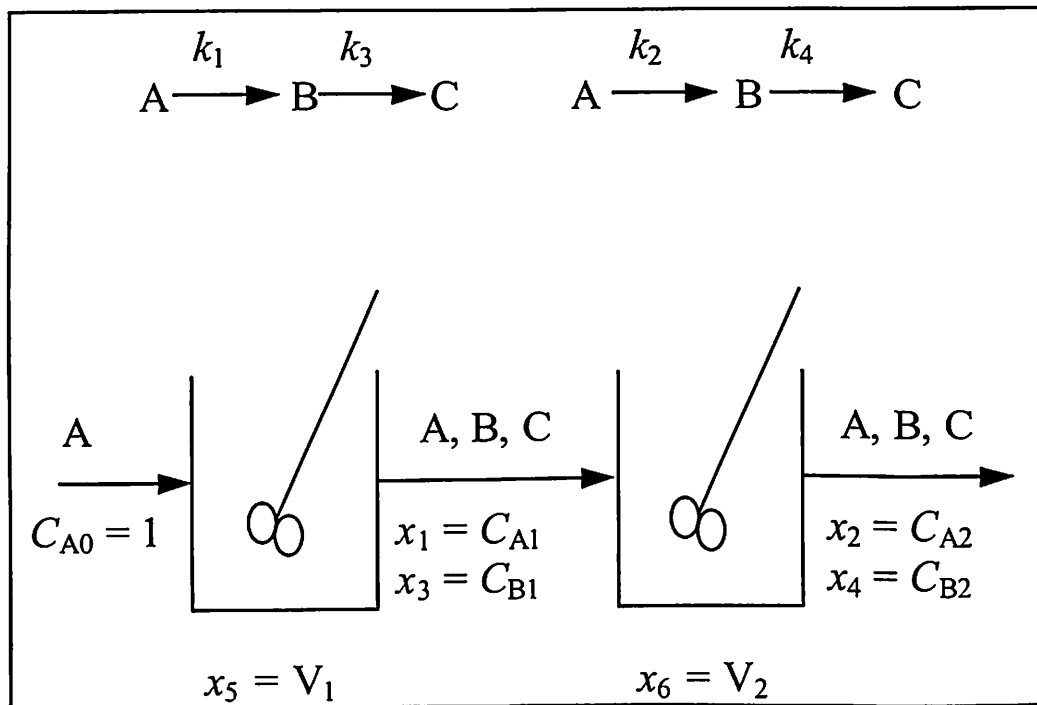


Fig. 3.30. Schematic of reactor network design problem

The global optimum is:  $(x_1, x_2, x_3, x_4, x_5, x_6; f) = (0.771462, 0.516997, 0.204234, 0.388812, 3.036504, 5.096052; -0.388812)$ .

Ryoo and Sahinidis (1995) solved the same problem using branch and reduce method which is based on the branch and bound method. A number of variations of the main algorithm were considered and the best one yielded a CPU of 21 s on a SPARC 2. Maranas and Floudas (1997) used geometric programming approach and problem gets converged to global optimum after 299 iterations and 20 s of CPU time.

$$x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 = 0$$

$$x_5^{0.5} + x_6^{0.5} \leq 4$$

$$(0, 0, 0, 0, 10^{-5}, 10^{-5}) \leq (x_1, x_2, x_3, x_4, x_5, x_6) \leq (1, 1, 1, 1, 16, 16).$$

Where  $k_1 = 0.09755988$

$$k_2 = 0.99k_1$$

$$k_3 = 0.0391908$$

$$k_4 = 0.9k_3$$

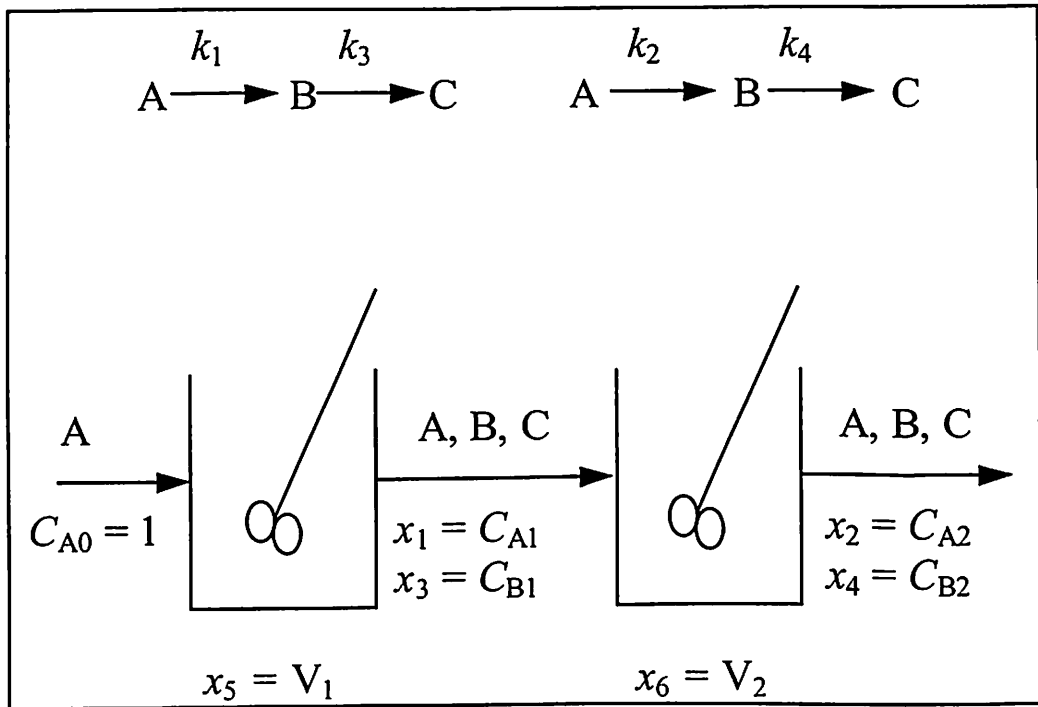


Fig. 3.30. Schematic of reactor network design problem

The global optimum is:  $(x_1, x_2, x_3, x_4, x_5, x_6; f) = (0.771462, 0.516997, 0.204234, 0.388812, 3.036504, 5.096052; -0.388812)$ .

Ryoo and Sahinidis (1995) solved the same problem using branch and reduce method which is based on the branch and bound method. A number of variations of the main algorithm were considered and the best one yielded a CPU of 21 s on a SPARC 2. Maranas and Floudas (1997) used geometric programming approach and problem gets converged to global optimum after 299 iterations and 20 s of CPU time.

This example constitute a very difficult test problem as it possesses a local minimum with an objective function value that is very close to that of the global solution. The local solutions are with  $f = -0.375$  and  $f = -0.3881$ . Interestingly enough, the two local solutions utilize only one of the two reactors whereas the global solution makes use of both reactors.

### 3.5.3.2. Problem Reformulation

This problem can be reformulated by eliminating equality constraint as follows:

$$\text{Max } f = \frac{k_2 x_6 (1 + k_3) + k_1 (1 + k_2 x_6)}{(1 + k_1 x_5)(1 + k_2 x_6)(1 + k_3 x_5)(1 + k_4 x_6)}$$

Subject to

$$x_5^{0.5} + x_6^{0.5} \leq 4$$

$(10^{-5}, 10^{-5}) \leq (x_5, x_6) \leq (16, 16)$ . Global optimum is same after reformulation. In the present study (Angira and Babu, 2003), the reformulated problem is solved using ~~DE & Deans~~ MDE algorithms.

### 3.5.3.3. Results and Discussion

Table-3.15 shows the results obtained using MDE (with/without forcing the bound) and its comparison to DE and  $\alpha$ BB algorithm. The key parameters used are  $CR = 0.8$ ,  $F = 0.5$  and  $NP = 10D$ .  $NFE$  and  $NRC$  represent respectively, the mean number of objective function evaluations and the percentage of runs converged to the global optimum in all the 100 experiments (with different seed values).

Termination criterion used is either (1) the maximum number of generations is reached (assumed 5000 generations), or (2)  $|f_{\max}^k - f_{\min}^k| < 10^{-6}$  where  $f$  is the value of objective function for  $k$ -th generation. Convergence tolerance in the present case is  $10^{-6}$  as compared to  $10^{-3}$  used by Adjiman et al. (1998b). RND is a difficult problem as mentioned already having two local optima near global optimum. Still MDE and

DE are able to locate the global optimum although the success rate (*NRC*) is 44 to 57% (Table-3.15). In this problem, *NFE* with forced bound method is 2.87% and 9.33% (for MDE and DE respectively) more than *NFE* for method without forcing the bound (Table-3.15). It is important to note that in this problem, *NRC* with forced bound method is just 8% and 10% respectively for MDE and DE as shown in Table-3.15. It is because when the upper limit of variable is violated, the value of variable is forced to the upper limit that resulted in convergence to non-optimal solution. This happened as one of the local solutions near to global optimum is lying on the bound, and hence trapped at local optimum.

The time taken by DE and MDE is less than that of  $\alpha$ BB algorithm. Of course the CPU-time cannot be compared directly because different computers are used. However, a comparison can be made after considering a factor of 10 (high enough), i.e., if the same problem would have been solved on HP9000/730 (66 MHz) using DE, it might have taken ten times more of CPU-time than by Pentium-III, 500 MHz. Even then the CPU-time is 92% less than that of  $\alpha$ BB algorithm.

**Table-3.15. Results of MDE<sup>#</sup> and its comparison with DE<sup>#</sup> and  $\alpha$ BB algorithms**

Methods	CPU-time	<i>NFE</i>	<i>NRC</i>
DE (FBM)	0.049**	1605	10
DE (MWFB)	0.041**	1468	57
MDE (FB)	0.041**	1289	08
MDE (MWFB)	0.034**	1253	44
$\alpha$ BB Algorithm	5.5* s	Not reported	Not reported

\* CPU-time obtained using HP9000/730 (66 MHz) with convergence tolerance of 0.001 (Adjiman et al., 1998b).

\*\* CPU-time obtained using Pentium-III (500 MHz) with convergence tolerance of 0.00001 (present study).

# CPU-time, *NFE*, *NRC* for DE are average of 100 experiments with different seeds.

The performance of MDE, as is evident from results presented in Table-3.15, is better than that of DE and  $\alpha$ BB algorithm. The reliability of DE and MDE is same as indicated by value of *NRC*, which is almost same for both the algorithms. RND is

difficult problem as mentioned already having two local optima near global optimum out of which one is within 0.2% of global optimum. It is so close to global optimum that a tolerance of more than 0.183% may lead to local optimum solution. Still MDE and DE are able to locate the global optimum 44 to 57 times out of 100 experiments.

Fig. 3.31 shows the convergence history of RND problem. Each point on the graph represents an average of 100 independent experiments. It is seen that error variation is more in case of MDE algorithm (0.003% to 0.014%) than that of DE (0.001% to 0.01%). This indicates that both MDE and DE are able to locate the optimum within  $\pm 0.014\%$  of global optimum. Also DE is found to be slightly more reliable as can be seen from success rate (*NRC*) values for the two algorithms (Table-3.15), though MDE is faster than DE (CPU-time is less for MDE as compared to DE as shown in Table-3.15).

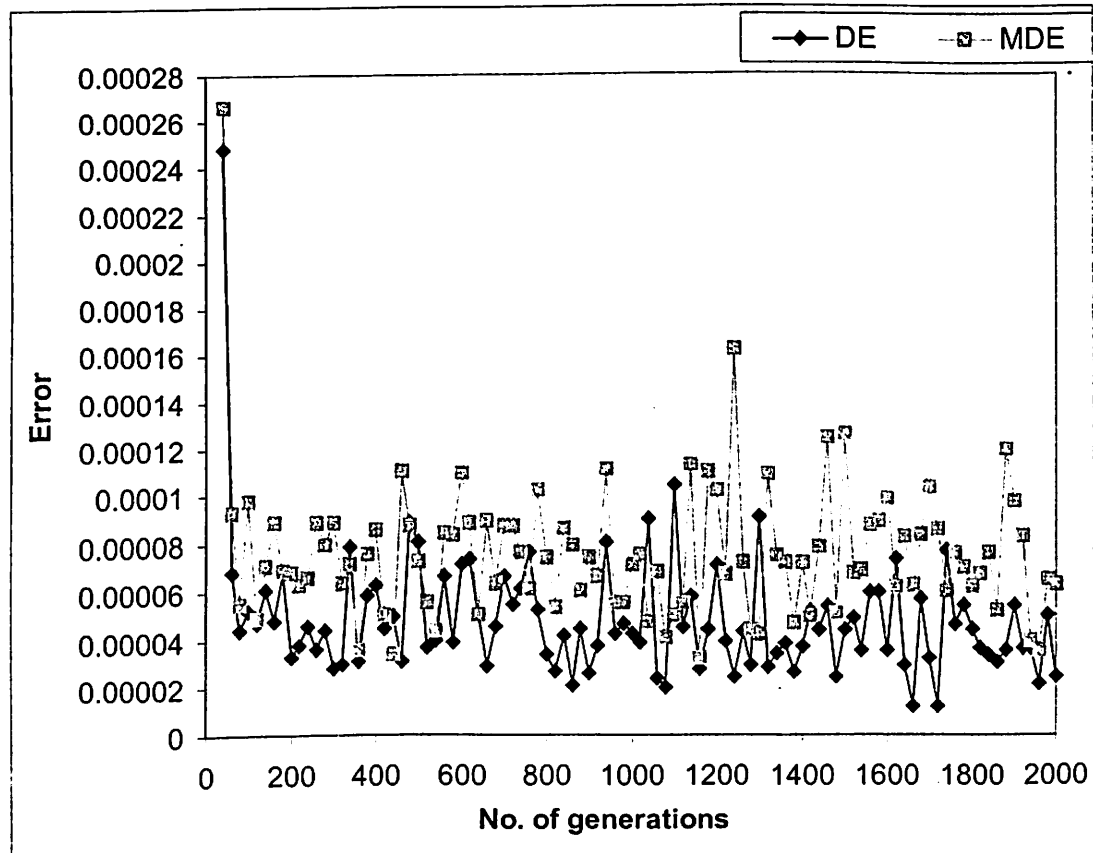


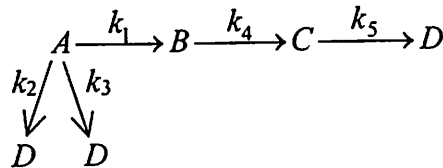
Fig. 3.31. Convergence history of RND problem

### 3.5.3.4. Conclusions

Results indicate that the performance of DE is better than that of  $\alpha$ BB algorithm. Also the MDE algorithm is found to take less CPU-time than that of DE and  $\alpha$ BB algorithms. Success rate is found to be slightly higher for DE than MDE algorithm. The error variation is found to be more in case MDE algorithm than that of DE. Both DE and MDE algorithms are able to locate the global optimum. The results obtained using DE and MDE algorithms matches with that reported in literature.

### 3.5.4. Isothermal CSTR Design Problems

For a reaction scheme shown below, a problem of finding the best isothermal yield of a product  $C$  in a CSTR is chosen. This problem is taken from Rosenbrock and Storey (1966). Later, Umeda and Ichikawa (1971) solved this problem using modified complex method. In this reaction scheme,  $A$  is the feed material,  $B$  is a transient intermediate,  $C$  is the required product, and  $D$  is an unwanted side product. The reaction scheme is as follows:



The rate constants are given by the equation,

$$k_i = C_i \exp \{-E_i/R (1/T - 1/658)\}$$

where  $C_i$  are frequency factors,  $E_i$  are activation energies (cal/mol),  $R$  is the gas constant (1.9872 cal/mol/K), and  $T$  is absolute temperature in K. The values of  $C_i$  and  $E_i$  for five rate constants are given in Table-3.16.

**Table-3.16. Values of  $C_i$  and  $E_i$**

	1	2	3	4	5
$C_i$	1.02	0.93	0.386	3.28	0.084
$E_i$	16000	14000	15000	10000	15000



### 3.5.4.1. Mathematical formulation

The kinetic equations for the reaction scheme shown above are given as follows:

$$\frac{dx_1}{dt} = -(k_1 + k_2 + k_3)x_1;$$

$$\frac{dx_2}{dt} = k_1x_1 - k_4x_2;$$

$$\frac{dx_3}{dt} = k_4x_2 - k_5x_3;$$

with the initial conditions  $x_1(0) = 1$ ,  $x_2(0) = 0$  and  $x_3(0) = 0$ ; where  $x_1$ ,  $x_2$ , and  $x_3$  are the concentration of material A, B, and C respectively. For constant temperature  $T$ , the kinetic equations stated above can be integrated analytically to give the following expression for C ( $f = x_3$ ).

$$x_3 = f = \left( \frac{k_1 k_4 e^{-k_3 t}}{k - k_4} \right) \left[ \left( \frac{1 - e^{-(k_4 - k_5)t}}{k_4 - k_5} \right) - \left( \frac{1 - e^{-(k - k_5)t}}{k - k_5} \right) \right]; \text{ Where } k = k_1 + k_2 + k_3;$$

Hence the maximizing function is given by

$$\text{Max. } x_3 = f$$

Subject to the following constraints,

$$0 \leq t \leq 10; \quad \text{Reaction time (s)}$$

$$200 \leq T \leq 2000; \quad \text{Reaction temperature (K)}$$

In this problem, the best isothermal yield of C is found by maximizing this expression with respect to the contact time ( $t$ ) and the temperature ( $T$ ). The results reported in literature are shown in Table-3.17.

**Table-3.17. Results reported in literature**

Methods	Yield of C ( $f = x_3$ )	Temperature ( $T$ )	Holding Time ( $t$ )
Hill climbing (Rosenbrock & Storey, 1966)	0.42308	978.96	0.0781
Modified Complex method (Umeda and Ichikawa, 1971)	0.42308	983.3	0.07572
Controlled Random Search (CRS) (Goulcher and Long, 1978)	0.4225	---	---

### 3.5.4.2. Results and Discussion

Rosenbrock and Storey (1966) used hill climbing method, Umeda and Ichikawa (1971) used modified complex method, and Goulcher and Long (1978) solved this problem using a Controlled Random Search (CRS) technique. None of them reported the global optimum solution for this problem. Reported values are shown in Table-3.17. In the current study, DE and MDE are able to locate a better solution (possibly the global optima) as DE and MDE proved to give global optimum for many test problems, as well as chemical engineering problems in this study.

**Table-3.18. Computational Results using DE and MDE for Isothermal Reactor**

Methods	CPU-time/ <i>NRC</i> / <i>NFE</i>		Yield of C ( $f = x_3$ )	Temperature ( $T$ )	Holding Time ( $t$ )
	FBM	MWFB			
DE*	0.033/100/8600	0.032/100/7996	0.423084	983.2028	0.075901
MDE*	0.027/100/7351	0.029/100/7685	0.423084	983.2028	0.075901

\*On Pentium-4/2.4GHz/256 MB (RAM)

<sup>§</sup> Calculated value using  $T = 983.3$  K and  $t = 0.07572$  s

The computational results are shown in Table-3.18 where *NFE*, and CPU-time are average of 100 experiments and *NRC* is percentage convergencies to global optimum solution. The optimal result obtained by hill climbing method (Rosenbrock and Storey, 1996) is  $x_3 = 0.42308$ ,  $T = 978.96$  and  $t = 0.0781$  at the 146<sup>th</sup> iteration for the initial point  $(T, t) = (1073.0, 0.5)$ . They found that it was not always possible to obtain the optimal point because of the existence of a ridge (Fig. 3.32). Fig. 3.32 shows the variation of objective function with temperature and time. Umeda and Ichikawa (1971) reported slightly higher value for temperature (983.3 K) and lesser value for holding time (0.07572 s) as compared to Rosenbrock and Storey (1996) using hill climbing method, although the value of the objective function is reported to be same in both the cases, i.e., 0.42308. However, as shown in Table-3.5.16, DE and MDE are able to find better value of optimum ( $f = x_3 = 0.423084$ , with temperature,  $T =$

983.2028, and the holding time,  $t = 0.075901$ ). DE takes 10.34% and 22.22% more CPU-time than that of MDE in case of method without forcing the bound and forced bound method respectively. Success rate ( $NRC$ ) is 100% for both DE and MDE algorithms.

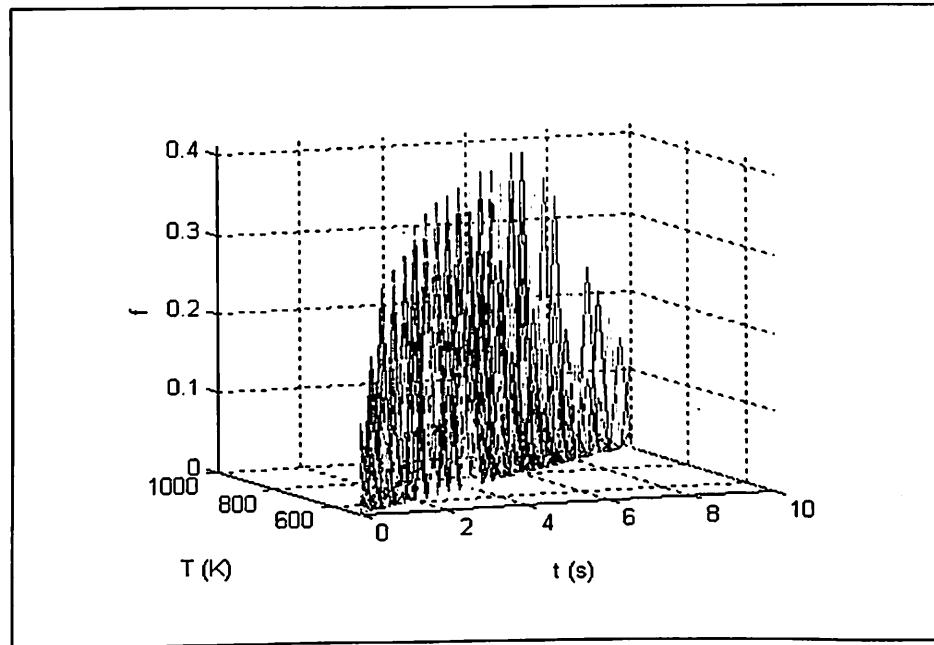


Fig. 3.32. Variation of  $f$  with time and temperature (temperature range: 500-1000 K)

Fig. 3.33 shows the convergence history of isothermal CSTR design problem. Each point on the graph represents the average value over 100 independent experiments. Though from naked eye, objective function value appears to be same for both DE and MDE after 60 generations but it is not so. After 82 generations the value of the objective function is 0.423074 for DE and 0.423084 for MDE while DE reaches a value of 0.423084 in 96<sup>th</sup> generation. This shows that MDE algorithm is able to locate global optimum faster than DE algorithm.

#### 3.5.4.3. Conclusions

The problem of isothermal CSTR design is solved using DE and MDE in the present section. Results indicate that the forced bound method takes slightly less (about 6.89%) CPU-time than the method without forcing the bound for MDE

algorithm while it is more (3.0%) for DE algorithm. Both MDE and DE algorithms are found to be able to locate the global optimum with 100% success rate. The performance of MDE is found to be the best in the problem studied.

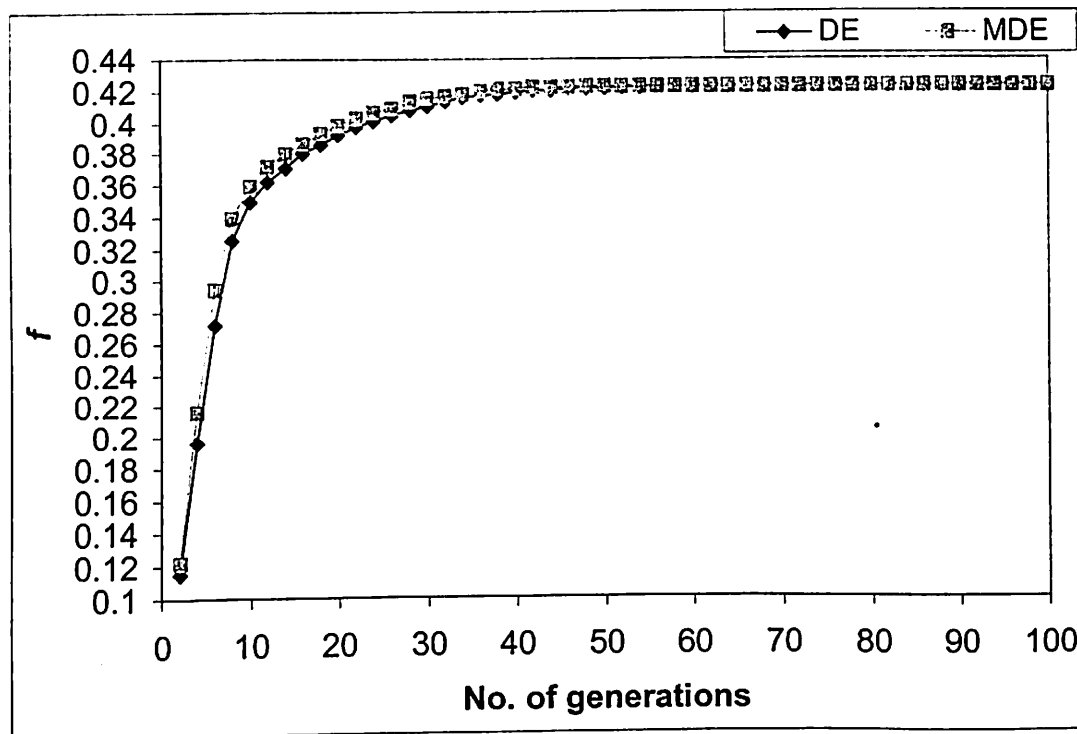


Fig. 3.33. Convergence history of isothermal CSTR design problem

### 3.5.5. Alkylation Process Optimization

#### 3.5.5.1. Introduction

The alkylation process is an important unit that is used in refineries to upgrade light olefins and isobutane into much more highly valued gasoline component. The light olefins are produced mainly from catalytic crackers and also from cokers and vis breakers. The primary alkylation reaction involves the reaction of isobutane with a light olefin, such as butylene, in the presence of strong acid catalyst to form the high octane, trimethyl pentane isomer. The alkylate obtained is one of the best gasoline blending components produced in the refinery because of its high octane, typically 96RON, and low vapor pressure which allows lower cost butane to be put into gasoline. The high profitability of upgrading light olefins and isobutane in the

refinery from LPG value to gasoline value explains why it is a very popular process alternative that is used at most locations that have catalytic crackers. Alkylation technology has been used for a long time in the refining industry. A simplified process flow diagram of an alkylation process is shown in Fig. 3.34.

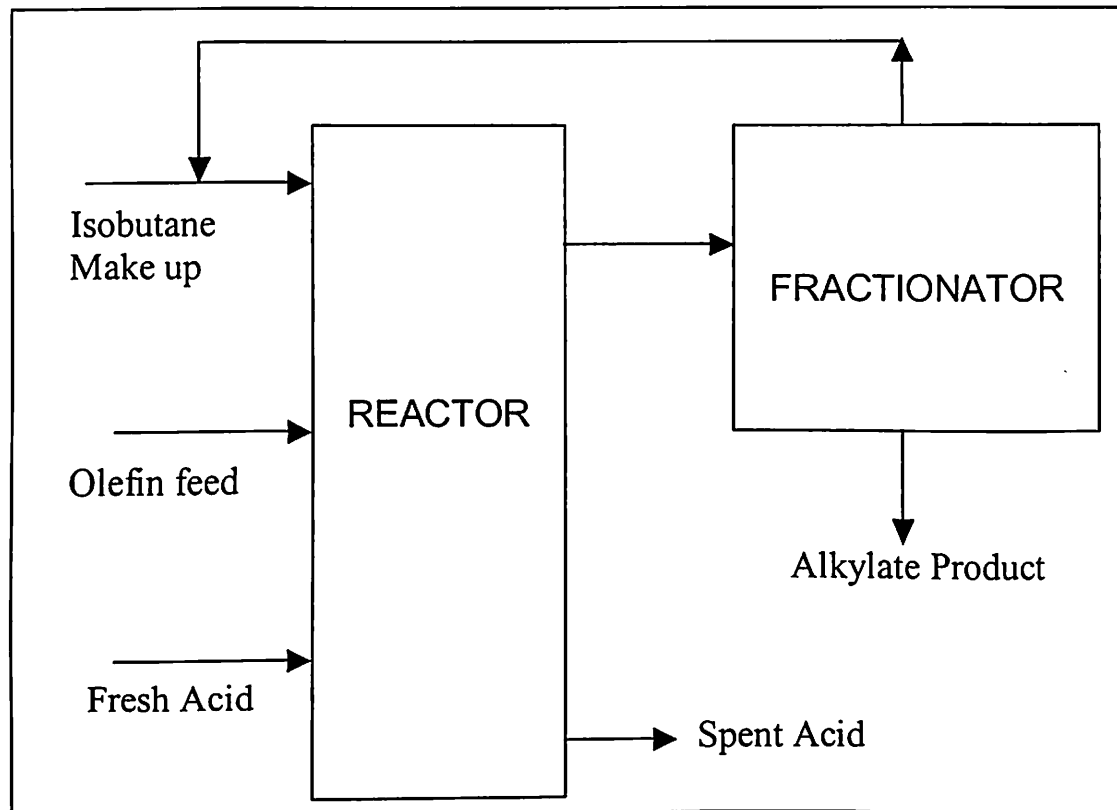


Fig. 3.34. Simplified alkylation process flow sheet

Alkylation process is common in the petroleum industry. The process model was described and solved by Sauer et al. (1964) using successive linear programming. The process model seeks to determine the optimum set of operating conditions for the process, based on a mathematical model, which allowed maximization of profit. Bracken and McCormick (1968) formulated the problem as a direct nonlinear programming model with mixed nonlinear inequality and equality constraints and a nonlinear profit function to be maximized. They used Sequential Unconstrained Minimization Technique (SUMT) for solving the same. Later, Dembo (1976) transformed the 10 variable NLP which Bracken and McCormick (1968) derived, into

seven variable problem. All equality constraints are eliminated and the problem has been formulated as a signomial optimization problem. This problem involves seven nonlinear variables subject to twelve nonlinear and two linear inequality constraints. Edger and Himmelblau (1989) used sequential quadratic programming to solve the problem as formulated by Bracken and McCormick (1968). Maranas and Floudas (1997) used generalized geometric programming to solve the seven variables problem as formulated by Dembo (1976). Adjiman et. al. (1998a) used  $\alpha$ BB algorithm (for general twice –differentiable constraint NLPs) for solving this problem. In the present study, the problem formulation is same as that of Maranas and Floudas (1997) and Adjiman et. al. (1998a). The problem is briefly discussed below.

### 3.5.5.2. *The Problem*

As shown in Fig. 3.34, an olefin feed (100% butane), a pure isobutene recycle and a 100% isobutene make-up stream are introduced in a reactor together with an acid catalyst. The reactor product stream is then passed through a fractionator where the isobutene and the alkylate product are separated. The spent acid is also removed from the reactor. The variables are defined as shown in Table-3.19 along with the upper and lower bounds on each variable. The bounds represent economic, physical and performance constraints.

**Table-3.19. Variables and their bounds**

Symbol	Variable	Lower Bound	Upper Bound
$x_1$	Olefin feed rate (barrels/Day)	1500	2000
$x_2$	Acid addition rate (thousands of pounds per day)	1	120
$x_3$	Alkylate yield (barrels/ day)	3000	3500
$x_4$	Acid strength (wt. %)	85	93
$x_5$	Motor octane no.	90	95
$x_6$	External Isobutane-to-olefin Ratio	3	12
$x_7$	F-4 performance no.	145	162

### 3.5.5.2.1. Profit Function

The objective is to improve the octane number of some olefin feed by reacting it with isobutene in the presence of acid. The product of the reaction is distilled and the un-reacted is recycled back to the reactor. The objective function was defined in terms of alkylate product, or output value minus feed and recycle costs. Operating costs were not reflected in the function. The total profit per day, to be maximized (Adjiman et al., 1998a), is given as follows:

$$\text{Max. Profit} = 1.715x_1 + 0.035x_1x_6 + 4.0565x_3 + 10.0x_2 - 0.063x_3x_5$$

Subject to:

$$0.0059553571x_6^2x_1 + 0.88392857x_3 - 0.1175625x_6x_1 - x_1 \leq 0,$$

$$1.1088x_1 + 0.1303533x_1x_6 - 0.0066033x_1x_6^2 - x_3 \leq 0,$$

$$6.66173269x_6^2 + 172.39878x_5 - 56.596669x_4 - 191.20592x_6 - 10000 \leq 0,$$

$$1.08702x_6 + 0.32175x_4 - 0.03762x_6^2 - x_5 + 56.85075 \leq 0,$$

$$0.006198x_7x_4x_3 + 2462.3121x_2 - 25.125634x_2x_4 - x_3x_4 \leq 0,$$

$$161.18996x_3x_4 + 5000.0x_2x_4 - 489510.0x_2 - x_3x_4x_7 \leq 0,$$

$$0.33x_7 - x_5 + 44.333333 \leq 0,$$

$$0.022556x_5 - 0.007595x_7 - 1.0 \leq 0,$$

$$0.00061x_3 - 0.0005x_1 - 1.0 \leq 0,$$

$$0.819672x_1 - x_3 + 0.819672 \leq 0,$$

$$24500.0x_2 - 250.0x_2x_4 - x_3x_4 \leq 0,$$

$$1020.4082x_4x_2 + 1.2244898x_3x_4 - 100000x_2 \leq 0,$$

$$6.25x_1x_6 + 6.25x_1 - 7.625x_3 - 100000 \leq 0,$$

$$1.22x_3 - x_6x_1 - x_1 + 1.0 \leq 0,$$

The maximum profit as reported in Adjiman et. al. (1998a) is: \$1772.77 per day, and the optimal variable values are  $x_1 = 1698.18$ ,  $x_2 = 53.66$ ,  $x_3 = 3031.3$ ,  $x_4 = 90.11$ ,  $x_5 = 95.0$ ,  $x_6 = 10.50$ ,  $x_7 = 153.53$ .

### 3.5.5.3. Results and Discussion

Table-3.20 presents the comparison of results obtained in earlier studies and those obtained using DE in present study. The optimal solution obtained using DE is:  $x_1 = 1698.256922$ ;  $x_2 = 54.274463$ ;  $x_3 = 3031.357313$ ;  $x_4 = 90.190233$ ;  $x_5 = 95.0$ ;  $x_6 = 10.504119$ ;  $x_7 = 153.535355$ ; and the value of objective function is 1766.36. This solution satisfies all the inequality constraints to six decimal places while solution reported by Maranas and Floudas (1997) and Adjiman et al. (1998a) violates the first, third and sixth constraints. The values of first, third and sixth constraints are found to be 0.016501, 4.752125, and 1727.867362 respectively instead of zero or less than zero. Hence the global optimal solution is 1766.36 that satisfy all the constraints to six decimal places (i.e., 0.0000001).

**Table-3.20. Comparison of Various Methods**

S. No.	Method	Profit (\$/day)	CPU-time (s)
1	SUMT (Bracken and McCormick, 1968)	1769	Not reported
2	NPSOL (Edgar and Himmelblau, 1989; 2001)	1768.75	Not reported
3	GGP in GAMS (Maranas and Floudas, 1997)	1773	30**
4	$\alpha$ BB algorithm (Adjiman et al., 1998a)	1772.77	13.6***
5	DE (Present study)	1766.36	5.12*

\* CPU-time on PC with Pentium PIII, 500 MHz/128 MB RAM/ 10 Gb HD

\*\* CPU-time on HP-730 workstation

\*\*\* HP9000/730, using scaled Gerschgorin theorem method

Table-3.21 shows the results obtained using DE and MDE in the present study (Angira and Babu, 2005b) and their comparison in terms of number of objective function evaluations, CPU-time and proportion of convergencies to the optimum. The



termination criterion used is an accuracy value of  $1 \times 10^{-6}$ . *NFE*, *NRC* and CPU-time in Table-3.21 represents, respectively the mean number of objective function evaluations over all the 100 experiments (with different seed values), the percentage of convergences to the global optimum, and the average CPU time per experiment (key parameters used are  $NP = 10D$ ,  $CR = 0.8$ ,  $F = 0.5$ ).

**Table-3.21. Results of DE and MDE for Alkylation problem**

Method <sup>#</sup>	<i>NFE</i>	<i>NRC</i>	CPU-time (s)*
DE (MWFB)	114895	100	5.81
MDE (MWFB)	108103	100	5.67
DE (FBM)	100126	100	5.12
MDE (FBM)	92287	100	4.77

<sup>#</sup> Penalty used is  $10^5$

\* CPU-time on PC with Pentium PIII, 500 MHz/128 MB RAM/ 10 Gb HD

Both DE and MDE are able to locate the better solution (possibly global optimum) in all the experiments, as *NRC* is 100%. MDE takes 2.47% less CPU-time than DE in case of method without forcing the bound, while it is 7.33% in case of forced bound method. This is due to the fact that some of the optimum values of decision variables are lying on the extreme, i.e., either on upper or on lower bound.

Fig. 3.35 shows the convergence history of alkylation problem using DE and MDE. After 160<sup>th</sup> generation, the value of the objective function is 1734.814 using MDE and 1719.956 using DE. This indicates that MDE is faster than DE and takes less CPU-time to locate the global optimum solution.

#### 3.5.5.4. Conclusions

The results obtained by the two methods (DE and MDE) satisfy the constraints to six decimal places as against the results reported by Adjiman et. al. (1998a) and Maranas and Floudas (1997). Both MDE and DE are able to find global optimum with 100% success rate. Also, the MDE is found to take less CPU-time as compared to DE indicating better performance of the algorithm than DE.

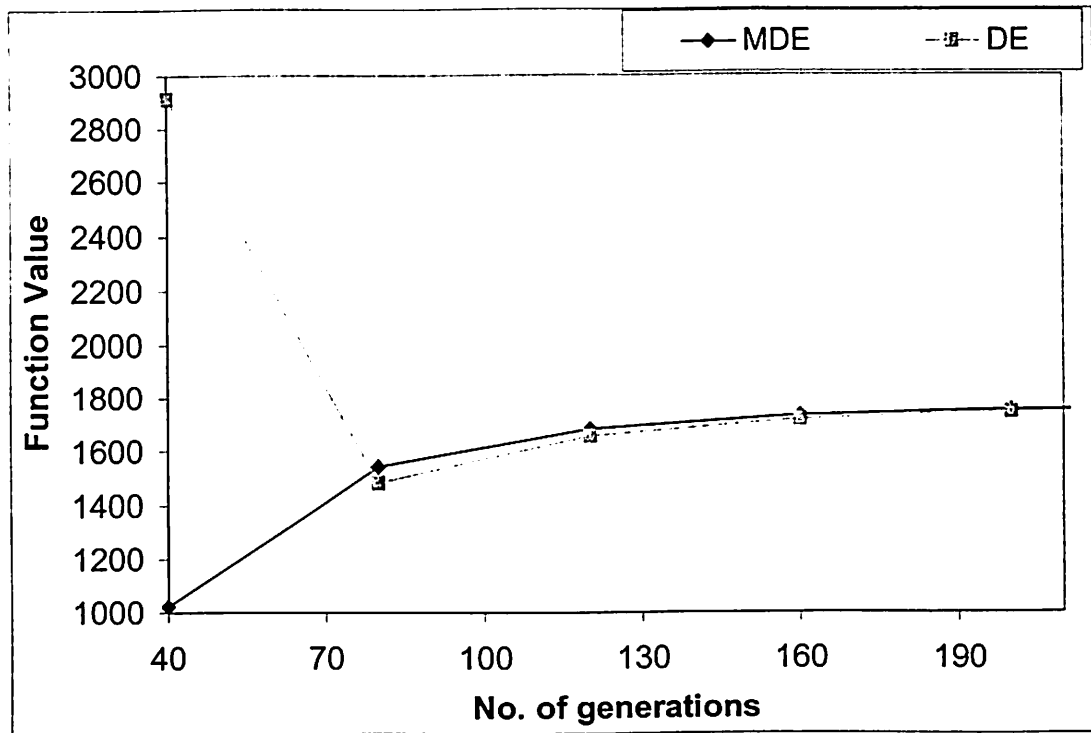


Fig. 3.35. Convergence history of alkylation problem

### 3.5.6. Optimization of Drying Process (DP)

#### 3.5.6.1. Introduction

In the optimization problem of a drying process for a through-circulation dryer (Chung, 1972), the objective is to find the air flow rate and the bed thickness which will maximize the production rate under certain constraints for a given material of known particle characteristics drying at a given temperature. An increase of the airflow rate increases the drying production rate at the expense of increased pressure drop and power consumption. An increase of the bed thickness also increases the pressure drop and the power consumption. Though an increase of the bed thickness decreases the drying rate, the net result may be an increase in the production rate (per pound of dry solid).

#### 3.5.6.2. Problem Formulation

The problem formulation is similar to that discussed by Chung (1972) and Luss and Jaakola (1973). The problem deals with the maximization of drying production

rate in a through-circulation dryer (Chung, 1972). The drying production rate, in terms of the independent operating variables, is a non-linear objective function, and is optimized under the nonlinear inequality constraint function using DE and MDE. We use the data and corrected equations as in Chung [1972; 1973]. The problem is to find the mass flow rate  $x_1$ , and the bed thickness  $x_2$  such that the drying production rate ( $P$ ) is maximized. Mathematically, it can be expressed as follows:

Maximize  $P$

$$P = 0.033x_1 \left[ \frac{0.036}{1 - e^{107.9x_2/x_1^{0.41}}} + 0.095 - B \right]^{-1}$$

$$\text{where, } B = \frac{9.27 \times 10^{-4} x_1^{0.41}}{x_2} \ln \left( \frac{1 - e^{-5.39x_2/x_1^{0.41}}}{1 - e^{-107.9x_2/x_1^{0.41}}} \right)$$

Subject to the following constraints:

(1) Power constraint

$$0.2 - 4.62 \times 10^{-10} x_1^{2.85} x_2 - 1.055 \times 10^{-4} x_1 \geq 0$$

(2) Pressure drop constraint

$$4/12 - 8.20 \times 10^{-7} x_1^{1.85} x_2 - 2.25/12 \geq 0.$$

(3) Drying time ratio constraint

$$0.64 - 109.6 \frac{x_2}{x_1^{0.41}} \left[ \frac{0.036}{1 - e^{107.9x_2/x_1^{0.41}}} + 0.095 - B \right] \geq 0$$

### 3.5.6.3. Results and Discussion

Chung [1973] solved the DP problem using differential algorithm and arrived at the maximum drying production rate ( $P$ ) of 172.5 lb/ft<sup>2</sup>hr, with  $x_1 = 975.6$  lb/ft<sup>2</sup>hr, and  $x_2 = 0.524$  ft. Luss and Jaakola [1973] reported the maximum  $P$  of 172.49 lb/ft<sup>2</sup>hr, with mass flow rate of 976.76 lb/ft<sup>2</sup>hr & bed thickness of 0.5235 ft. Goulcher and Long (1978) used CRS method to solve this problem and obtained maximum drying production rate of 172.46 lb/ft<sup>2</sup>hr.

In Table-3.22 and Table-3.23, *NFE* & *NRC* represent respectively, the mean number of objective function evaluations and the percentage of experiments converged to the global optimum in all the 100 experiments (with different seed values). The key parameters used are  $F = 0.7$ ;  $CR = 0.99$ . The stopping criteria adopted for DE is to terminate the search process when one of the following conditions is satisfied: (1) the maximum number of generations is reached (assumed 1000 generations. (2)  $|f_{\max}^k - f_{\min}^k| < 10^{-4}$  where  $f$  is the value of objective function for  $k$ -th generation.

Table-3.22 shows the results obtained using DE (FBM) in the present study (Babu and Angira, 2002a) and its comparison with traditional direct search method and CRS method. They cannot be compared on the basis of CPU-time since the computer used is different in the each case. However the value of objective function, i.e., the performance index is slightly better using DE as compared to direct search method and controlled random search. But the *NFE* is slightly less in case of direct search method (7.35% less than that of DE). Also, it is interesting to note here that the value of  $F$  has a significant effect on convergence to optimal solution. It is found that DE did not converge to the same optimal solution using different seeds for  $F$  values less than 0.6. However, the  $CR$  (the crossover constant) has very little effect on optimal solution.

**Table-3.22. Comparison of DE with DSM and CRS methods (DP problem)**

Method	Performance Index	( <i>NFE</i> /CPU-time)
DE	172.49	1428/0.065 <sup>5</sup> sec
DSM	172.47	1323/5* sec
CRS	172.46	Not reported/0.269 <sup>7</sup> sec

<sup>5</sup> Pentium III/500 MHz, and strategy is DE/rand/1/bin

\*IBM 370/165

<sup>7</sup>CDC 7600

Results of MDE are compared with that of DE in Table-3.23. *NRC* is almost same in both DE and MDE. This indicates that both the techniques are equally reliable. Also a high value of *NRC* in case of without forcing the bound indicates superiority of this method handling bound violation. The *NFE* using method without forcing the bound is almost same as that with forced bound method for the two algorithms (Table-3.23) while there is significant difference in *NRC* value. It is 85% and 58% for method without forcing the bound and forced bound method respectively in case of MDE algorithm. Although there is not much difference in *NRC* value for the two algorithms using either forced bound method or method without forcing the bound. MDE takes almost 10.0% less time than DE for both methods, i.e., method without forcing the bound and forced bound method. Also, MDE takes less *NFE* as compared to DSM (Table-3.22).

**Table-3.23. Results of MDE and DE algorithms for DP problem**

Methods	<i>NRC/NFE/CPU-time</i> <sup>§</sup> (FBM)	<i>NRC/NFE/CPU-time</i> <sup>§</sup> (MWFB)
DE	54/1415/0.065	79/1428/0.065
MDE	58/1238/0.059	85/1255/0.058

§ Pentium III/500 MHz

Fig. 3.36 shows the convergence history of the drying problem. Each point on the graph represents the average of 100 independent experiments. Fig. 3.36 clearly indicates that MDE algorithm is faster than DE. After 30 generations, the value of objective function (Performance Index) is found to be 172.364 and 172.076 for MDE and DE algorithms respectively. MDE algorithm took 50 generations as compared to 60 generations for DE to locate the global optimum solution.

#### 3.5.6.4. Conclusions

In the present section, the problem of optimization of drying process for a through-circulation dryer is solved using MDE and DE. Also the results are compared with

DSM and CRS methods. Results indicate that the DE and MDE are able to find a slightly better objective function value than that of CRS and DSM. Method without forcing the bound is found to give high success rate (*NRC*) as compared to forced bound method, for both the algorithms, as the optimum solution does not lie on the boundary (i.e., lower or upper limit). Although the *NFE* or CPU-time is same for both forced bound method and method without forcing the bound. It is found that the performance of MDE is better than that of DE and traditional direct search method in finding the true global optima for nonconvex nonlinear problem.

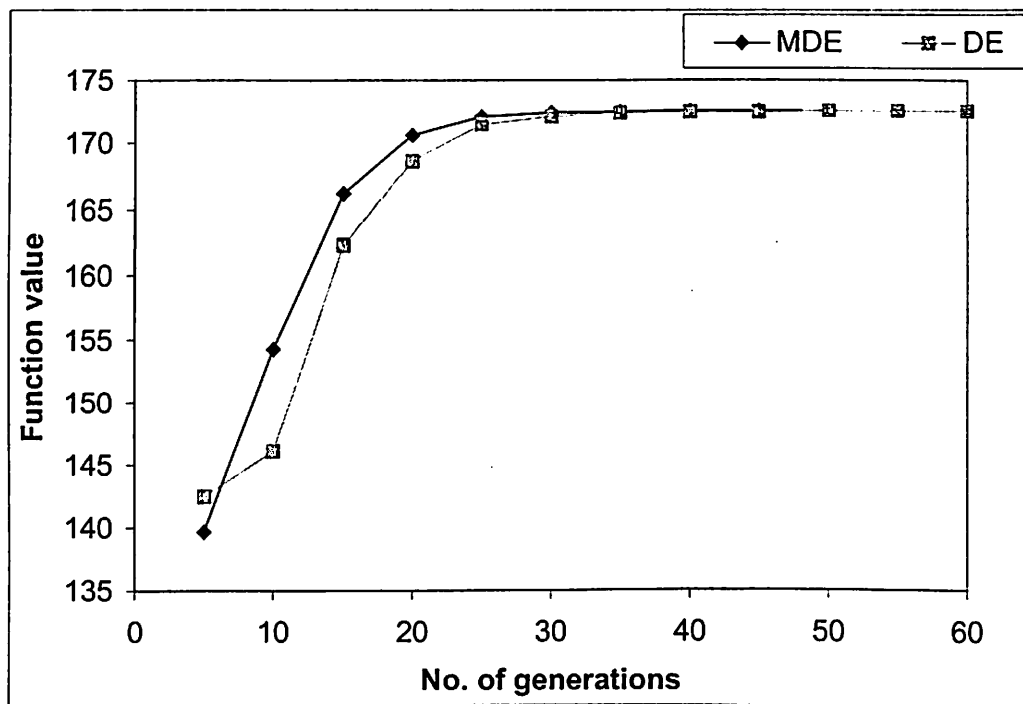


Fig. 3.36. Convergence history of drying process problem

### 3.5.7. Optimum Fuel Allocation in Power Plant (FAPP)

#### 3.5.7.1. Introduction

This problem deals with a power plant with two boiler-turbine-generator combinations (Fig. 3.37), each capable of using a mixture of two types of fuels (fuel oil and fuel gas). Each of two boiler-turbine-generator combinations can be fired with either fuel oil or fuel gas or any combination of the two fuel types. It is desired to select the fuel mix for each generator so as to minimize the fuel oil utilization. In the

present section, the problem of optimum fuel allocation in power plants is solved using DE and MDE evolutionary computation methods. Comparison is made with a direct search procedure (which utilizes pseudo random numbers over a region) and controlled random search method.

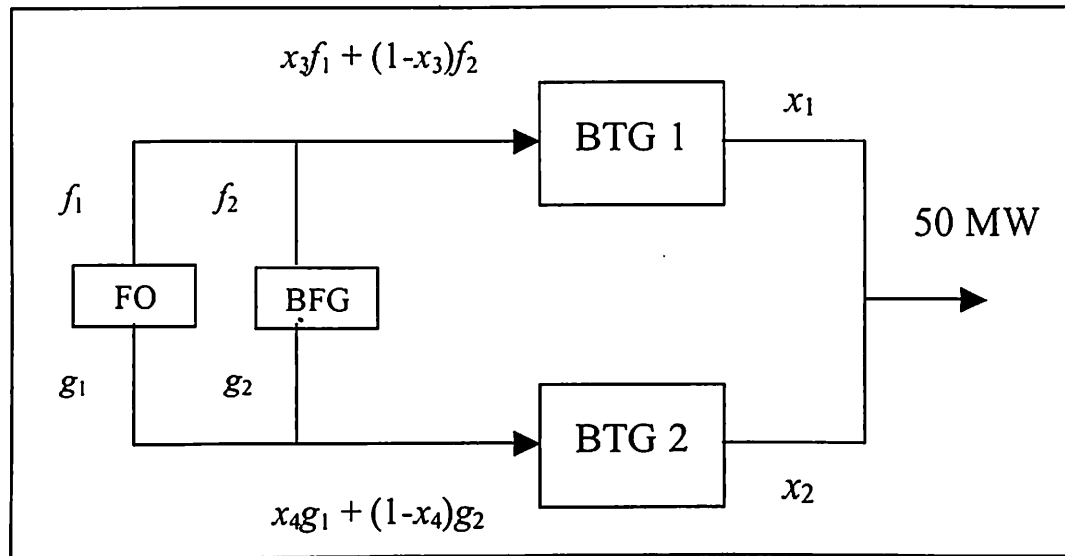


Fig. 3.37. Schematic of two-boiler-turbine-generator power plant

### 3.5.7.2. The Problem

The problem of minimizing the purchase of fuel oil (FO) is considered. In a power plant it is desired to produce an output of 50 MW from a two boiler-turbine-generator combination (Fig. 3.37). It can use fuel oil or blast furnace gas (BFG) or any combination of these. The maximum BFG that is available is specified. The fuel consumption-load characteristics are nonlinear. By applying nonlinear curve-fitting Hovanessian and Stout (1963) obtained the fuel requirements for the two generators explicitly in terms of MW produced. For generator-1 the fuel requirements for fuel oil in tons per hour is given by the equation (3.17):

$$f_1 = 1.4609 + 0.15186x_1 + 0.00145x_1^2 \quad (3.17)$$

and for BFG in fuel units per hour

$$f_2 = 1.5742 + 0.1631x_1 + 0.001358x_1^2 \quad (3.18)$$

where  $x_1$  is the output in MW of generator-1.

Similarly for generator-2 the fuel oil requirement is:

$$g_1 = 0.8008 + 0.2031x_2 + 0.000916x_2^2 \quad (3.19)$$

and for BFG,

$$g_2 = 0.7266 + 0.2256x_2 + 0.000778x_2^2 \quad (3.20)$$

where  $x_2$  is the output in MW of generator-2.

### 3.5.7.2.1. Problem Formulation

The following assumptions are made:

1. Only 10.0 units of BFG are available per hour.
2. Each generator may use any combination of fuel oil or BFG.
3. When a combination of fuel oil and BFG is used the effects are additive, i.e., if in generator-1 we use fuel oil and BFG in 1/3 ratio to produce  $x_1$  MW, then the total fuel consumption consists of  $0.25 f_1$  tons of fuel oil per hour and  $0.75 f_2$  fuel units of BFG per hour.

The problem is to produce 50 MW from the two generators in such a way that the amount of fuel oil consumed is minimum. Mathematically, the problem can be formulated as follows:

$$\text{Minimize } C = x_3 f_1 + x_4 g_1. \quad (3.21)$$

where  $f_1$  and  $g_1$  are given by equations (3.17) and (3.19) respectively.

Subject to the following constraints:

- (a) Operating range for the generator-1

$$18 \leq x_1 \leq 30 \quad (3.22)$$

- (b) Requirement of 50 MW of Power

$$x_2 = 50 - x_1 \quad (3.23)$$

- (c) Operating range of generator-2



$$14 \leq x_2 \leq 25 \quad (3.24)$$

(d) Fraction of fuel oil used in generator-1

$$0 \leq x_3 \leq 1 \quad (3.25)$$

(e) Fraction of fuel oil used in generator-2

$$0 \leq x_4 \leq 1 \quad (3.26)$$

(f) Availability of blast furnace gas (BFG)

$$\text{BFG} = (1 - x_3) f_2 + (1 - x_4) g_2 \leq 10.0 \quad (3.27)$$

where  $f_2$  and  $g_2$  are given by Equation (3.18) and (3.20) respectively.

Hence the problem is to choose the variables  $x_1$ ,  $x_3$ , and  $x_4$  so that  $C$  as given by equation (3.21) is minimized because the variable  $x_2$  is eliminated by using equation (3.23). There are five inequality constraints embodied in equation (3.22) and equations (3.24) to (3.27). Also note that there is no lower limit restriction on equation (3.27) since computationally BFG cannot become negative.

### 3.5.7.3. *Results and Discussion*

For FAPP problem, Hovanessian and Stout (1963) and Hovanessian and Pipes (1969) obtained the minimum fuel oil consumption of 3.17 tons/hour by using separable programming where the nonlinearities were approximated by linear sections and the problem was solved by the standard linear programming procedure. Luss and Jaakola (1973) obtained the minimum fuel consumption of 3.05 tons/hour by using the optimization procedure based on direct search and systematic search region reduction. Goulcher and Long (1978) used CRS method to solve this problem and obtained minimum fuel consumption of 3.065 tons/hour.

In Table-3.24 and Table-3.25, *NFE* & *NRC* represent respectively, the mean number of objective function evaluations and the percentage of experiments converged to the global optimum in all the 100 experiments (with different seed

values). The key parameters used are  $F = 0.7$ ;  $CR = 0.99$ . The stopping criteria adopted for DE is to terminate the search process when one of the following conditions is satisfied: (1) the maximum number of generations is reached (assumed 1000 generations). (2)  $|f_{\max}^k - f_{\min}^k| < 10^{-4}$  where  $f$  is the value of objective function for  $k$ -th generation.

**Table-3.24. Comparison of DE with DSM and CRS methods (FAPP problem)**

Methods	Performance Index	(NFE/CPU-time)
DE	3.0521	3418/0.11 <sup>5</sup> sec
DSM	3.0526	2989/1* sec
CRS	3.065	Not reported/0.276 <sup>#</sup> sec

<sup>5</sup> Pentium III/500 MHz, and strategy is DE/rand/1/bin

\*IBM 370/165

<sup>#</sup>CDC 7600

Table-3.24 shows the results obtained using DE (FBM) in the present study (Babu and Angira, 2002a) and its comparison with traditional direct search method and CRS methods. They cannot be compared on the basis of CPU-time since the computer used is different in the each case. However the value of objective function, i.e., the performance index is slightly better using DE as compared to direct search and controlled random search methods. But the *NFE* is slightly less in case of direct search method (12.55 % less than that of DE).

**Table-3.25. Results of MDE and DE algorithms for FAPP problem**

Methods	NRC/NFE/CPU-time <sup>5</sup>	
	FBM	MWFB
DE	96/3418/0.11	82/4198/0.140
MDE	97/3079/0.101	83/3830/0.126

<sup>5</sup> Pentium III/500 MHz

Results obtained using MDE algorithm are compared with that of DE in Table-3.25. *NRC* is almost same in both DE and MDE algorithms. This indicates that both the techniques are equally reliable. Also *NRC* is found to be more in case of forced bound method as compared to method without forcing the bound in both the

values). The key parameters used are  $F = 0.7$ ;  $CR = 0.99$ . The stopping criteria adopted for DE is to terminate the search process when one of the following conditions is satisfied: (1) the maximum number of generations is reached (assumed 1000 generations). (2)  $|f_{\max}^k - f_{\min}^k| < 10^{-4}$  where  $f$  is the value of objective function for  $k$ -th generation.

**Table-3.24. Comparison of DE with DSM and CRS methods (FAPP problem)**

Methods	Performance Index	(NFE/CPU-time)
DE	3.0521	3418/0.11 <sup>5</sup> sec
DSM	3.0526	2989/1* sec
CRS	3.065	Not reported/0.276 <sup>#</sup> sec

<sup>5</sup> Pentium III/500 MHz, and strategy is DE/rand/1/bin

\*IBM 370/165

<sup>#</sup>CDC 7600

Table-3.24 shows the results obtained using DE (FBM) in the present study (Babu and Angira, 2002a) and its comparison with traditional direct search method and CRS methods. They cannot be compared on the basis of CPU-time since the computer used is different in the each case. However the value of objective function, i.e., the performance index is slightly better using DE as compared to direct search and controlled random search methods. But the *NFE* is slightly less in case of direct search method (12.55 % less than that of DE).

**Table-3.25. Results of MDE and DE algorithms for FAPP problem**

Methods	NRC/NFE/CPU-time <sup>5</sup>	
	FBM	MWFB
DE	96/3418/0.11	82/4198/0.140
MDE	97/3079/0.101	83/3830/0.126

<sup>5</sup> Pentium III/500 MHz

Results obtained using MDE algorithm are compared with that of DE in Table-3.25. *NRC* is almost same in both DE and MDE algorithms. This indicates that both the techniques are equally reliable. Also *NRC* is found to be more in case of forced bound method as compared to method without forcing the bound in both the

algorithms (DE and MDE). *NRC* using MDE is 83% for method without forcing the bound and 97% for forced bound method. The *NFE* using method without forcing the bound is slightly more than with forced bound method (Table-3.25) in case of both the algorithms. MDE takes almost 27.85% less computational time than that using DE (using MWFB).

Fig. 3.38 shows the convergence history of the fuel allocation problem. Each point on the graph represents the average of 100 independent experiments. Fig. 3.38 clearly indicates that MDE algorithm is faster than DE. After 44 generations, the value of objective function is found to be 3.070293 and 3.080094 for MDE and DE algorithms respectively. MDE algorithm took 108 generations as compared to 124 generations for DE to locate the global optimum solution.

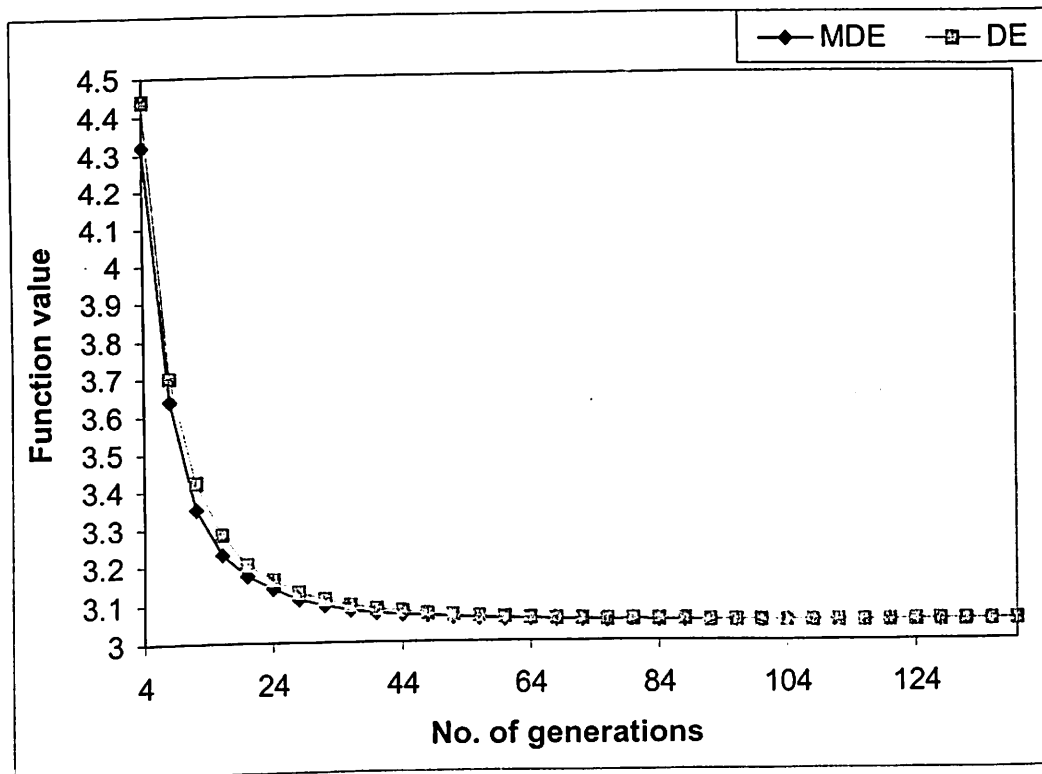


Fig. 3.38. Convergence history of FAPP problem

#### 3.5.7.4. Conclusions

A slightly better value of objective function is found using DE and MDE algorithms as compared to DSM and CRS. Also, results indicate that MDE algorithm

is able to locate the global optimum faster than DE. Forced bound method is found to give high success rate (*NRC*) and less *NFE*, i.e., CPU-time as compared to without forcing, for both the algorithms, as one of the variable occupies the boundary value (i.e., lower limit). The performance of MDE is found to be better than that of DE.

### 3.5.8. Water Pumping System

#### 3.5.8.1. Introduction

Optimization of fluid flow systems encompasses a wide-ranging scope of problems. Many optimization problems exist in which the process model represent flow through a single pipe, flow in parallel pipes, compressors, heat exchangers, and so on. In the present section, DE and MDE algorithms are used to solve the classical optimization problem of water pumping system. Comparison is made with Branch & Reduce algorithm. The results indicate that performance of DE and MDE are better than that of the Branch & Reduce algorithm.

#### 3.5.8.2. The Problem

A water pumping system (Stoecker, 1971) consists of two parallel pumps drawing water from a lower reservoir and delivering it to another that is 40 m higher, as shown in Fig. 3.39. In addition to overcoming the pressure difference due to the elevation, the friction in the pipe is  $7.2w^2$  kPa, where  $w$  is the combined flow rate in kilograms per second. The pressure-flow-rate characteristics of the pumps are:

$$\text{Pump 1:} \quad \Delta p \text{ (kPa)} = 810 - 25w_1 - 3.75w_1^2$$

$$\text{Pump 2:} \quad \Delta p \text{ (kPa)} = 900 - 65w_2 - 30w_2^2$$

where  $w_1$  and  $w_2$  are the flow rates through pump-1 and pump-2, respectively.

The system can be represented by four simultaneous equations. The pressure difference due to elevation and friction is:

$$\Delta p = 7.2w^2 + \frac{(40 \text{ m})(1000 \text{ kg/m}^3)(9.807 \text{ m/s}^2)}{1000 \text{ Pa/kPa}} \quad (3.28)$$

$$\text{Pump-1:} \quad \Delta p = 810 - 25w_1 - 3.75w_1^2 \quad (3.29)$$

$$\text{Pump-2:} \quad \Delta p = 900 - 65w_2 - 30w_2^2 \quad (3.30)$$

$$\text{Mass balance:} \quad w = w_1 + w_2 \quad (3.31)$$

The objective here is to minimize  $\Delta p$  subject to the constraints (3.28), (3.29), (3.30), and (3.31).

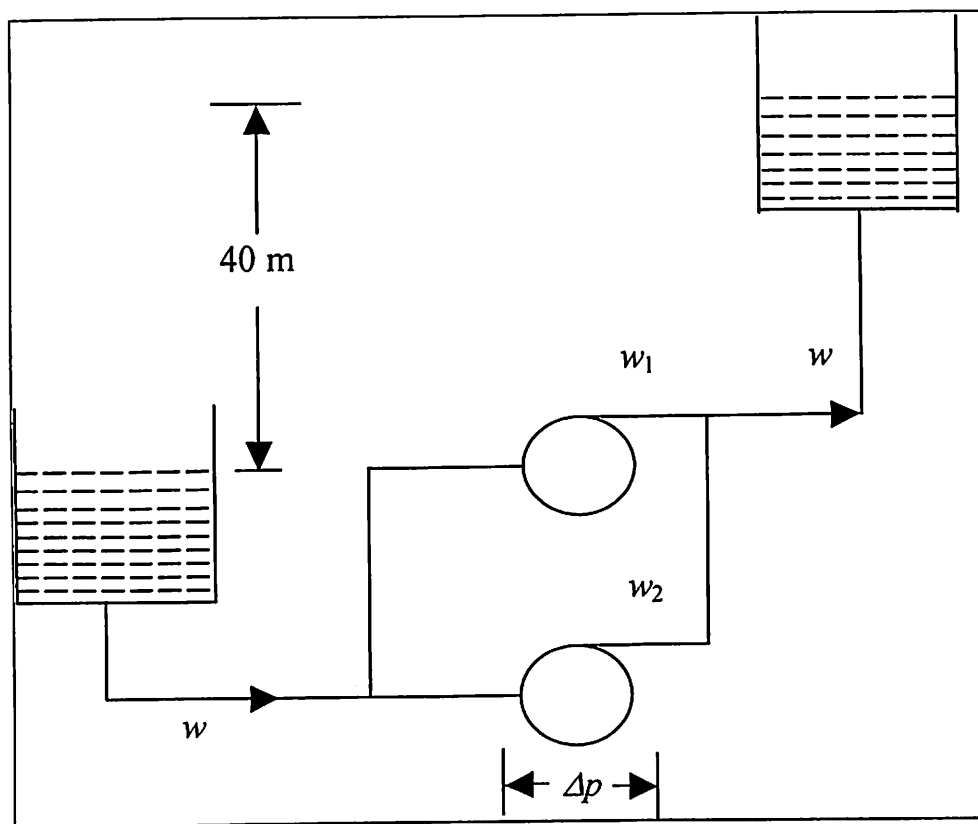


Fig. 3.39. Schematic of water pumping system

Hence the problem can be stated as follows:

$$\text{Min. } \Delta p = 7.2w^2 + \frac{(40 \text{ m})(1000 \text{ kg/m}^3)(9.807 \text{ m/s}^2)}{1000 \text{ Pa/kPa}}$$

Subject to the following constraints:

$$\Delta p = 810 - 25w_1 - 3.75w_1^2.$$

$$\Delta p = 900 - 65w_2 - 30w_2^2.$$

$$w = w_1 + w_2.$$

Stoecker (1971) used the method of successive substitution for solving this problem.

### 3.5.8.2.1. Problem Modification

Liebman et al. (1986) modified the above problem as given below:

$$\text{Min. } f = x_3,$$

Subject to the following constraints:

$$x_3 = 250 + 30x_1 - 6x_1^2$$

$$x_3 = 300 + 20x_2 - 12x_2^2$$

$$x_3 = 150 + 0.5(x_1 + x_2)^2$$

$$0 \leq x \leq (9.422, 5.903, 267.42).$$

Ryoo and Sahinidis (1995) solved this problem using Branch and Reduce algorithm. They used different strategies of Branch and Reduce algorithm. The CPU-time reported by them ranges from a minimum of 0.3 s to a maximum of 150 s for various strategies used by them on Sun SPARC station 2. However Ryoo and Sahinidis (1995) did not mention about configuration of the system used but from site <http://www.jinr.dubna.su/unixinfo/sun/suns1.html>, the available configuration for Sun SPARC station 2 is: CPU - (1x 40MHz), RAM - 32MB and Hard Disk - 1.2Gb. The termination criterion used was an accuracy ( $\epsilon$ ) =  $10^{-6}$ . The global optimum reported is  $(x; f) = (6.293429, 3.821839, 201.159334; 201.159334)$ .

### 3.5.8.2.2. Problem Reformulation-1

In general, the equality constraints are difficult to deal with. And more so in the case of evolutionary algorithms. So there is a need to transform equality constraints

into inequality constraints by some means or the other. Typically, they are handled by either of the following two methods, viz., (1) eliminating the parameter and hence reducing the dimensions of the problem (2) an equality constraint is formulated into two inequalities by introducing deviation variables on problem parameter. In the present study (Babu and Angira, 2003a), one variable is eliminated using method-1 while the other two equalities are transformed into inequalities using method-2. Hence, the reformulated problem is as follows:

$$\text{Min. } f = x_3 = 150 + 0.5(x_1 + x_2)^2$$

Subject to the following constraints:

$$6x_1^2 - 30x_1 - 249.9999999 + 150.0 + 0.5(x_1 + x_2)^2 \geq 0.0$$

$$12x_2^2 - 20x_2 - 299.9999999 + 150.0 + 0.5(x_1 + x_2)^2 \geq 0.0$$

$$0 \leq x \leq (9.422, 5.903)$$

The global optimum obtained is:  $(x; f) = (6.293429, 3.821839; 201.159334)$ .

### 3.5.8.3. Results and Discussion

Table-3.26 shows the results obtained by both DE and Branch & Reduce algorithm (Ryoo and Sahinidis, 1995). It may be noted that the global optimum is same as reported in Ryoo and Sahinidis (1995), i.e.,  $f = 201.159334$  and the flow rates through Pump-1 & Pump-2 are  $x_1 = 6.293430$  and  $x_2 = 3.821839$  respectively. With two of the three strategies they used, the CPU-time is 0.3s and with the third it is 150s (as shown in Table-3.26).

Table-3.27 and Table-3.28 presents the comparison, in terms of the number of objective function evaluations, CPU-time and proportion of convergencies to the optimum, between the different DE strategies. The termination criterion used is accuracy of  $10^{-6}$  and  $10^{-7}$  respectively. In these tables, *NFE*, *NRC* and CPU-time represents, respectively the mean number of objective function evaluations over all



the 10 experiments, the percentage of convergencies to the global optimum and the average CPU time per experiment. The key parameters used are:  $NP = 20$ ,  $CR = 0.5$ ,  $F = 0.8$ .

**Table-3.26. Comparison of DE with Branch & Reduce method**

Parameters	DE	Branch & Reduce
$(x_1)$	6.293430	6.293429
$(x_2)$	3.821839	3.821839
$(x_3) = f$ (i.e. same as the objective function)	201.159334	201.159334
CPU-time (s)	0.0714*	(0.3 – 150s) <sup>5</sup>
Objective function ( $f$ )	201.159334	201.159334

\* CPU-time on PC with Pentium PIII, 500 MHz/128 MB RAM/ 10 Gb HD using DE/rand-to-best/1/exp

<sup>5</sup> CPU-time on Sun SPARC Station 2

**Table-3.27. Results of DE with all ten strategies (accuracy ( $\epsilon$ )=  $10^{-6}$ )**

S. No.	Strategy	<i>NFE</i>	CPU- time (s)	<i>NRC</i> (%)
1	DE/rand/1/bin	3134	0.1319	100
2	DE/best/1/bin	2406	0.0879	100
3	DE/best/2/bin	4444	0.1758	100
4	DE/rand/2/bin	4644	0.1758	100
5	DE/rand-to-best/1/bin	2364	0.0879	100
6	DE/rand/1/exp	3214	0.1154	100
7	DE/best/1/exp	2372	0.0934	100
8	DE/best/2/exp	4506	0.1648	100
9	DE/rand/2/exp	4652	0.1868	100
10	DE/rand-to-best/1/exp	2162	0.0714	100

**Table-3.28. Results of DE with all ten strategies [accuracy ( $\epsilon$ )=  $10^{-7}$ ]**

S. No.	Strategy	<i>NFE</i>	CPU- time (s)	<i>NRC</i> (%)
1	DE/rand/1/bin	3524	0.1374	100
2	DE/best/1/bin	2624	0.1099	100
3	DE/best/2/bin	5016	0.1868	100
4	DE/rand/2/bin	5158	0.2033	100
5	DE/rand-to-best/1/bin	4146	0.1648	90
6	DE/rand/1/exp	3542	0.1319	100
7	DE/best/1/exp	2636	0.0989	100
8	DE/best/2/exp	5048	0.1923	100
9	DE/rand/2/exp	5206	0.1978	100
10	DE/rand-to-best/1/exp	2484	0.0989	100

### 3.5.8.3.1. Problem Reformulation-2

In the problem reformulation-1, first equality is used as true objective function while the other two equalities are converted into inequalities and treated as inequality constraints. Here it is again reformulated as unconstrained optimization problem, incorporating constraints into objective function, as follows:

$$\text{Min. } f = x_3 = 150 + 0.5(x_1 + x_2)^2 + R(h^2 + g^2)$$

Where

$$h = 6x_1^2 - 30x_1 - 250.0 + 150.0 + 0.5(x_1 + x_2)^2 = 0.0 ;$$

$$g = 12x_2^2 - 20x_2 - 300.0 + 150.0 + 0.5(x_1 + x_2)^2 = 0.0 ; \text{ and } R (=10^{10}) \text{ is penalty on constraint violation. } 0 \leq x \leq (9.422, 5.903).$$

The global optimum obtained is:  $(x; f) = (6.293429, 3.821839; 201.159334)$ . The problem is solved using DE and MDE (with or without forcing the bound). The key parameters used are:  $NP = 40$ ,  $CR = 0.8$ ,  $F = 0.5$  and the termination criterion used is  $\epsilon = 10^{-6}$ . Total 100 experiments are carried out. Mean value of  $NFE$ ,  $NRC$  and CPU-time is shown in Table-3.29 and Table-3.30 for method without forcing the bound and forced bound method respectively.

**Table-3.29. Results of DE and MDE using MWFB**

Methods	<i>NFE</i>	<i>NRC (%)</i>	CPU-time (s)
DE	2014	100	0.061
MDE	1630	100	0.047

**Table-3.30. Results of DE and MDE using FBM**

Methods	<i>NFE</i>	<i>NRC (%)</i>	CPU-time (s)
DE	2054	100	0.063
MDE	1651	100	0.047

It is interesting to note that from Table-3.29 and Table-3.30 that *NRC* and *NFE* are almost same in both the cases, i.e., for method without forcing the bound and forced bound method. However, CPU-time is less (about 25.4%) in case of MDE algorithm than that of DE.

#### **3.5.8.4. Conclusions**

In this section, the problem of optimization of water pumping system using DE and MDE algorithms has been solved. The time taken by DE is much less than that of Branch & Reduce algorithm (Table-3.26). Of course the CPU-times cannot be compared directly because different computers are used. From the above Table-3.27 and Table-3.28 it is evident that the strategy number-10 (DE/rand-to-best/1/exp) is the best strategy. It takes least average CPU-time, maximum *NRC* and minimum *NFE*. The results obtained by three methods (viz. MDE, DE and Branch & Reduce algorithm) are same and matches with that reported in literature. Performance of MDE is found to be better than DE and branch and reduce algorithm.

### **3.5.9. Liquid Extraction Problem**

#### **3.5.9.1. Introduction**

Liquid-liquid extraction is a mass transfer operation in which a liquid solution (the feed) is contacted with an immiscible or nearly immiscible liquid (solvent) that exhibits preferential affinity or selectivity towards one or more of the components in the feed. Two streams result from this contact: the extract, which is the solvent rich solution containing the desired extracted solute, and the raffinate, the residual feed solution containing little solute. Liquid-Liquid Extraction (LLE) is a powerful separation technique and in situations where distillation is not feasible for reasons such as a complex process sequence, high investment or operating costs, heat sensitive materials, or low volatility, extraction is often the best technology to use.

LLE is carried out either (1) in a series of well-mixed vessels or stages (well mixed tanks or plate columns or (2) in a continuous process such as spray columns, packed columns, and rotating disk columns. This example illustrates the application of Evolutionary Computation method such as DE and MDE to the optimization of a LLE system represented by a plug flow model. Steady state continuous countercurrent liquid extraction can be modeled in a variety of ways, the most common of which are (1) a plug flow model and (2) an axial dispersion model. Jackson and Agnew (1980) demonstrated the effectiveness of an on-line model based steady-state optimization scheme in finding and holding the optimum operation conditions of a liquid extraction pilot plant in which acetic acid was extracted from amyl alcohol using water as the solvent. The equipment could be operated either manually or under computer control. Under automatic operation, the computer could maintain the interface level, feed and solvent flow rates, and the stirrer speed ( $N_S$ ) to their respective set points. The latter three set points could be changed by the optimization routine. The interface level was controlled via the extract flow. The measured variables, for both control and optimization purposes were the feed and solvent flow rates; the feed, raffinate and extract concentrations; the stirrer speed; the interface position; and the feed temperature. Further details are given in Jackson (1977).

The process operation was subject to upper and lower limits on feed and solvent phase superficial velocities and the stirrer speed, and to minimum throughput and flooding constraints. For use in an optimization scheme, a process model is required to be able to predict the steady-state process output for a given set of inputs. These predicted values can then be used to calculate the value of the performance function. To be of use in an on-line scheme, the model must be amenable to solution without excessive computational effort. They examined the accuracy of four models viz., a

model based on plug flow of both phases, one based on axial dispersion superimposed on the flow of both phases, and two empirical models (one linear and one nonlinear) for a continuous pilot-scale extraction column in which water was used to extract acetic acid from amyl alcohol. The linear and non-linear models were direct correlations of experimentally obtained mass transfer data.

Two empirical models were rejected, as they were not fitting the data. Both plug-flow and axial-diffusion models enabled the correct optimum to be predicted in all cases, the differences in performance when these models were used were negligible, as there was little axial mixing of the phases in the Rotating Disc Contactor. Also, the marginally superior performance of the axial diffusion model was offset by the greater complexity of its solution compared to the plug flow model. Hence, the plug flow model offered the best compromise between predictive ability and complexity because of its greater simplicity. The same is used in the present study too. Once a model is specified, it can be used to determine the maximum extraction rate. A typical column is shown in Fig. 3.40 (where the internal rotating disc are not shown). The process model, objective function, and the constraints, are described in the following sections.

### 3.5.9.2. *Process Model*

Assuming that the concentrations are expressed on a solute free mole basis and that the equilibrium relation between  $Y$  and  $X$  is a straight, i.e., the phases are insoluble. The model is then given as below:

$$\frac{dX}{dZ} - N_{OX} (X - Y) = 0 \quad (3.32)$$

$$\frac{dY}{dZ} - F_E N_{OX} (X - Y) = 0 \quad (3.33)$$

where,

- $F_E$  = extraction factor ( $mv_x/v_y$ )  
 $m$  = distribution coefficient ( $m = 1.5$ )  
 $N_{OX}$  = number of transfer units  
 $v_x, v_y$  = Superficial velocity in raffinate, extract phase  
 $X$  = dimensionless raffinate phase concentration  
 $Y$  = dimensionless extract phase concentration  
 $Z$  = dimensionless contactor length

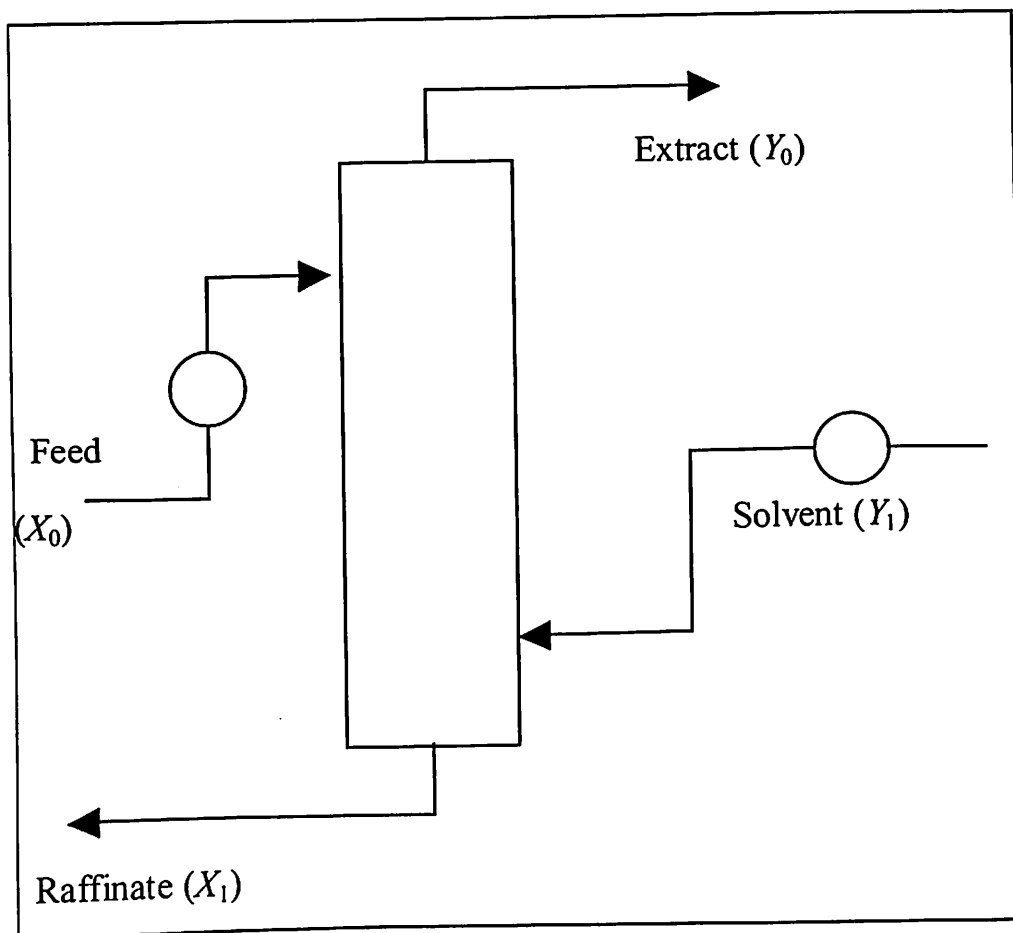


Fig. 3.40. Schematic of extraction column

Fig. 3.40 shows the extraction column with boundary conditions  $X_0$  and  $Y_1$ . A solution for  $Y_0$  in terms of  $v_x$  and  $v_y$  can be obtained, given the values for  $m$ ,  $N_{OX}$ , and the length of the column. Hartland and Mecklenburgh (1975) list the solution for the

plug flow model (and also the axial dispersion model) for a linear equilibrium relationship, in terms of  $F$ :

$$Y_0 = \frac{F_E [1 - \exp\{N_{OX}(1 - F_E)\}]}{1 - F_E \exp[N_{OX}(1 - F_E)]} \quad (3.34)$$

For the plug flow and axial diffusion models, Jackson and Agnew (1980) summarized a number of correlations for  $N_{OX}$  given by an equation of the form:

$$N_{OX} = a \left( \frac{v_X}{v_Y} \right)^b (N_S)^c \quad (3.35)$$

The correlations obtained by non-linear least-square regression were:

$$N_{OX} = 1.7 \left( \frac{v_X}{v_Y} \right)^{0.24} (N_S)^{0.069} \quad (3.36a)$$

for  $2.08 \leq N_S \leq 4.2s^{-1}$

$$N_{OX} = 0.2 \left( \frac{v_X}{v_Y} \right)^{0.24} (N_S)^{1.5} \quad (3.36b)$$

for  $4.2 \leq N_S \leq 8.33s^{-1}$

$$N_{OX} = 0.18 \left( \frac{v_X}{v_Y} \right)^{0.18} (N_S)^{0.088} \quad (3.36c)$$

for  $2.08 \leq N_S \leq 4.3s^{-1}$

$$N_{OX} = 0.09 \left( \frac{v_X}{v_Y} \right)^{0.24} (N_S)^{2.1} \quad (3.36d)$$

for  $4.3 \leq N_S \leq 8.33s^{-1}$

In the present study (Babu and Angira, 2003b), equation (3.36b) is used with  $N_S = 8.33s^{-1}$ .

$$N_{ox} = 4.81 \left( \frac{v_x}{v_y} \right)^{0.24} \quad (3.37)$$

### 3.5.9.2.1. Objective function

We have used the same objective function as proposed in Jackson and Agnew (1980), i.e., to maximize the total extraction rate for constant disk rotation speed subject to the inequality constraints:

$$\text{Maximize } f = v_y Y_0 \quad (3.38)$$

Inequality constraints:

Implicit constraints exist because of the use of dimensionless variables

$$X_0 \leq X \leq X_1$$

$$Y_1 \leq Y \leq Y_0 \quad (3.39)$$

Constraints on  $v_x$  and  $v_y$  would be upper and lower bounds such as:

$$0.05 < v_x < 0.25$$

$$0.05 < v_y < 0.30 \quad (3.40)$$

and flooding constraint is:

$$v_x + v_y \leq 0.20$$

### 3.5.9.3. Results and Discussion

Jackson and Agnew (1980) used a modified gradient-projection technique for linearly constrained optimization problems developed by Jackson (1977). Edgar & Himmelblau (1989) used GRG (generalized reduced gradient method) and obtained the same optimum (0.15, 0.05) as Jackson and Agnew (1980). The value of objective function is 0.225, i.e., the true optimum lay on the flooding constraint. In the present section, DE and MDE, the evolutionary techniques, are applied. The stopping criterion adopted is to terminate the search process when one of the following conditions is satisfied: (1) the maximum number of generations is reached (assumed



2000 generations). (2)  $|f_{\max}^k - f_{\min}^k| < 10^{-5}$  where  $f$  is the value of objective function for  $k$ -th generation.

Table-3.31 and Table-3.32 show the results obtained using MDE and DE (with or without forcing the bound). *NFE*, *NRC* and CPU-time are the number of function evaluations, number of experiments converged to global optimum and CPU-time in second respectively. All values in Table-3.31 and Table-3.32 are average of successful experiments only out of total 100 experiments. The key parameters used are:  $NP = 10 \cdot D$ ,  $CR = 0.8$ ,  $F = 0.5$ .

**Table-3.31. Results of DE and MDE for liquid extraction problem (MWFB)**

Method	<i>NFE</i>	<i>NRC</i> (%)	CPU-time (s)
DE	1496	82	0.054
MDE	1267	89	0.040

**Table-3.32. Results of DE and MDE for liquid extraction problem (FBM)**

Method	<i>NFE</i>	<i>NRC</i> (%)	CPU-time (s)
DE	856	98	0.027
MDE	724	99	0.024

From Table-3.31 and Table-3.32, it is evident that in case of forced bound method, *NFE* is less as compared to method without forcing the bound. Similarly, *NRC* is also high in case of forced bound method. This is because of optimum being found on extreme, i.e., on the bound. Also CPU-time is less (about 50%) in case of forced bound method than that of method without forcing the bound using both DE and MDE. However, MDE takes less CPU-time (about 26% and 11.1% respectively for MWFB and FBM) as compared to DE.

#### 3.5.9.4. Conclusions

In this section, the problem of optimization of liquid extraction process has been presented and solved using DE and MDE algorithms. MDE is found to be better than

DE in terms of CPU-time required to reach the global optima. Also, *NRC* is high using MDE as compared to DE for both, i.e., method without forcing the bound and forced bound method. When comparing DE and MDE, the performance of MDE is found to be better both in terms of *NRC* and *NFE*.

### 3.5.10. Heat Exchanger Network Design (HEND)

#### 3.5.10.1. Introduction

Heat integration brings about significant savings when the right balance between capital investment and operating costs is found. Thus heat exchanger network problems attracted attention for many years. In this section, DE and MDE have been used to solve the non-linear chemical engineering problem, i.e., heat exchanger network design. This problem represents difficult non-linear optimization problem, with equality and inequality constraints.

#### 3.5.10.2. Problem Formulation

This problem addresses the design of a heat exchanger network as shown in Fig. 3.41. It has been taken from Floudas and Pardalos (1990). Also, it is solved by Adjiman et. al. (1998b) using  $\alpha$ BB algorithm. One cold stream must be heated from 100 to 500 °F using three hot streams with different inlet temperatures. The goal is to minimize the overall heat exchange area.

$$\text{Min. } f = x_1 + x_2 + x_3$$

Subject to the following constraints:

$$0.0025(x_4 + x_6) - 1 = 0$$

$$0.0025(-x_4 + x_5 + x_7) - 1 = 0$$

$$0.01(-x_5 + x_8) - 1 = 0$$

$$100x_1 - x_1x_6 + 833.33252x_4 - 83333.333 \leq 0$$

$$x_2x_4 - x_2x_7 - 1250x_4 + 1250x_5 \leq 0$$

$$x_3x_5 - x_3x_8 - 2500x_5 + 1250000 \leq 0$$

$$100 \leq x_1 \leq 10000$$

$$1000 \leq x_2, x_3 \leq 10000$$

$$10 \leq x_4, x_5, x_6, x_7, x_8 \leq 1000$$

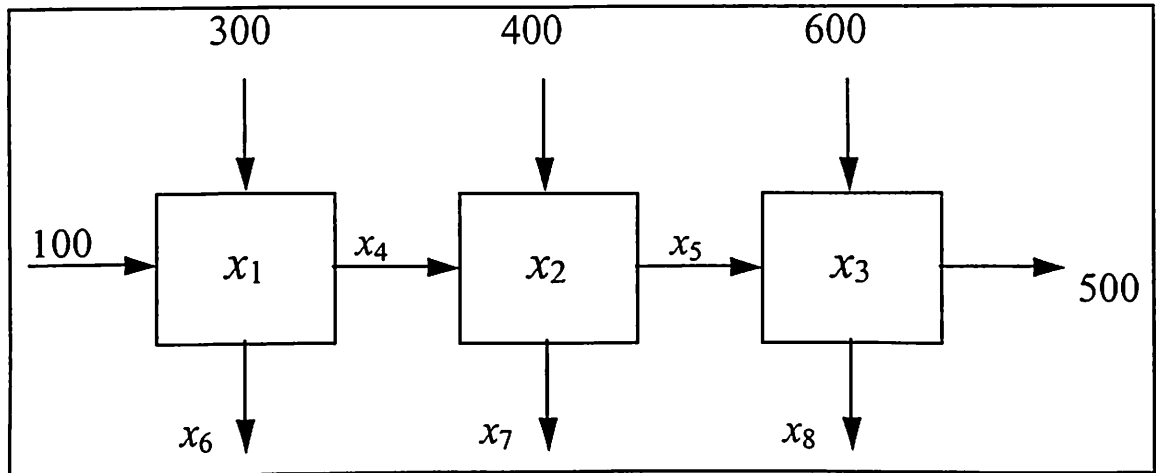


Fig. 3.41. Schematic of heat exchanger network design problem

The global optimum is:  $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, f) = (579.19, 1360.13, 5109.92, 182.01, 295.60, 217.9, 286.40, 395.60; 7049.25)$ .

The above problem can be reformulated by eliminating equality constraint as given below:

$$\text{Min } f = x_1 + x_2 + x_3$$

Subject to

$$100x_1 - x_1(400 - x_4) + 833.33252x_4 - 83333.333 \leq 0$$

$$x_2x_4 - x_2(400 - x_5 + x_4) - 1250x_4 + 1250x_5 \leq 0$$

$$x_3x_5 - x_3(100 + x_5) - 2500x_5 + 1250000 \leq 0$$

$$100 \leq x_1 \leq 10000$$

$$1000 \leq x_2, x_3 \leq 10000$$

$$10 \leq x_4, x_5 \leq 1000; \quad \text{Global optimum is same after reformulation.}$$

### 3.5.10.3. Results and Discussion

Table-3.33 shows the results obtained using DE (with/without forcing the bound) and its comparison to MDE and  $\alpha$ BB algorithm. The key parameters used are  $CR = 0.8$ ,  $F = 0.5$  and  $NP = 10D$ .  $NFE$ ,  $NRC$ , and CPU-time represent the mean value of the 100 experiments (with different seed values). Termination criterion used is either (1) the maximum number of generations is reached (assumed 5000 generations), (2)  $|f_{\max}^k - f_{\min}^k| < 10^{-5}$  where  $f$  is the value of objective function for  $k$ -th generation. Convergence tolerance in the present study (Angira and Babu, 2003) is  $10^{-5}$  as compared to  $10^{-3}$  used by Adjiman et. al. (1998b).

**Table-3.33. Comparison of DE with MDE and  $\alpha$ BB algorithm (HEND problem)**

Method	CPU-time	$NFE$	$NRC$
DE (FBM)	1.513**	38824	89
DE (MWFB)	1.477**	37810	100
MDE (FBM)	1.292**	33146	88
MDE (MWFB)	1.236**	31877	100
$\alpha$ BB Algorithm	54.4* s	Not reported	Not reported

\* CPU-time obtained using HP9000/730 (66 MHz) with convergence tolerance of 0.001 (Adjiman et al., 1998b).

\*\* CPU-time obtained using Pentium-III (500 MHz) with convergence tolerance of 0.00001 (present study).

Both MDE and DE algorithms are able to locate the global optimum although the success rate is 88 to 100% (Table-3.33). MDE takes about 16.3% less CPU-time (in method without forcing the bound) than that of DE. In this problem, CPU-time in forced bound method is 4.53% and 2.44% (for MDE and DE respectively) more than CPU-time for method without forcing the bound (Table-3.33). It is important to note that  $NRC$  with forced bound method is just 88% and 89% respectively for MDE and DE while it is 100% for both in case of method without forcing the bound. It is because when the upper limit of variable is violated, the value of variable is forced to the upper limit that resulted in convergence to non-optimal solution. The performance

of MDE, as is evident from results presented in Table-3.33, is better than that of DE and  $\alpha$ BB algorithm. The reliability of DE and MDE is same as indicated by *NRC*, which is same for both the algorithms.

The CPU-time taken by DE and MDE algorithms is less than that of  $\alpha$ BB algorithm. Of course the CPU-time cannot be compared directly because different computers are used. However, a comparison can be made after considering a factor of 10 (high enough), i.e., if the same problem would have been solved on HP9000/730 (66 MHz) using DE, it might have taken ten times more of CPU-time than at Pentium-III, 500 MHz. Even then the CPU-time is about 77.3% less using MDE (MWFB) and about 72.85% less using DE (MWFB) than that of  $\alpha$ BB algorithm.

Fig. 3.42 shows the convergence history of HEND problem. Each point on the graph represents an average of 100 independent experiments. It is seen that error reduces faster in case of MDE algorithm than that of DE. But both MDE and DE algorithms are able to locate the global optimum.

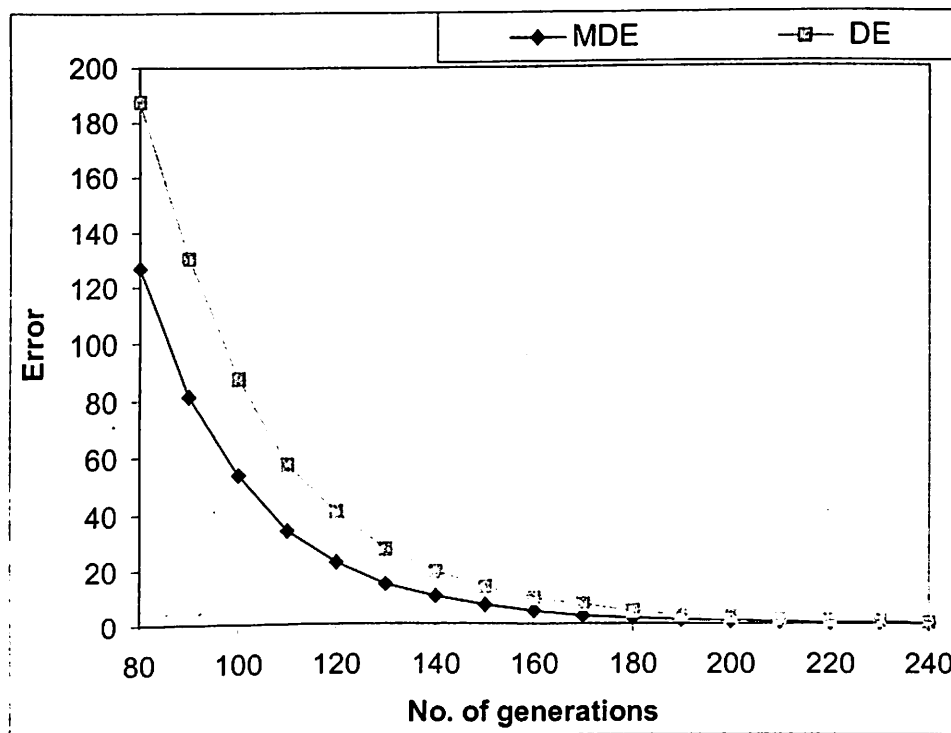


Fig. 3.42. Convergence history of HEND problem

The error obtained after 200 generations is 0.986 for MDE algorithm while it is 2.344 for DE. MDE is able to locate the global optimum for the HEND problem faster than DE algorithm. Also MDE is found to be reliable as can be seen from success rate (*NRC*) values for the two algorithms (Table-3.33).

#### **3.5.10.4. Conclusions**

Results indicate that the performance of DE is better than that of  $\alpha$ BB algorithm. Both DE and MDE algorithms are able to locate the global optimum with 100% success rate (*NRC*). Also the MDE algorithm is found to take less CPU-time than DE and  $\alpha$ BB algorithms. Results indicate that MDE is reliable, efficient and hence a better approach to the optimization of non-linear problems.

## APPLICATION OF MDE TO PROCESS SYNTHESIS AND DESIGN

### 4.1. Introduction

Process synthesis can be defined as the selection, arrangement, and operation of processing units so as to create an optimal scheme. In other words, it is an act of determining the optimal interconnection of processing units as well as the optimal type and design of units within a process system. The interconnection of processing units is called the structure of the process system. When the performance of the system is specified, the structure of the system and the performance of the processing units are not determined uniquely. This task is combinatorial and open-ended in nature and has received a great deal of attention over the past twenty-five years (Nishida et al., 1981). Since the synthesis problem is open ended, it has lead to the development of quite different approaches such as thermodynamic targets (Linnhoff, 1981), heuristic (Rudd et al., 1973; Douglas, 1988), evolutionary methods (Stephanopoulos and Westerberg, 1976), and optimization techniques (Grossmann,

1985). The present study deals with the structural flow sheet optimization problem that arises in the latter approach.

## 4.2. Background

The use of mathematical programming techniques for process synthesis has received considerable attention over the last two decades. For example, nonlinear programming (NLP) technique for heat exchanger networks (Floudas et al, 1986), and mixed integer nonlinear programming (MINLP) models for structural flowsheet optimization (Kocis and Grossmann, 1987, 1988, 1989; Floudas et al., 1989) to name a few. The major reason for this increased interest lies in the fact that mathematical programming techniques provide a systematic framework for process synthesis. Also, there has been substantial progress in methods and software for solving optimization problems, development of powerful modeling languages (General Algebraic Modeling System, GAMS), and technological advances in computing.

In order to formulate the synthesis problem as a mathematical programming problem, a superstructure is postulated which includes many alternate designs from which the optimal process will be selected. Once the superstructure is specified, the next task is to determine the optimal process flow sheet through structural and parameter optimization of the superstructure (which requires the solution of a mixed integer optimization problem). In early 1980s, most of the process synthesis and design problems have been formulated as Mixed-Integer Linear Programming (MILP) problems. Although these formulations (e.g. Papoulias and Grossmann, 1983) have proved to be quite powerful, they have the limitation that nonlinearities in the process equations cannot be treated explicitly and approximated through the discretization. The need for the explicit handling of the nonlinearities in the synthesis problem



motivates the use of Mixed-Integer Nonlinear Programming (MINLP). MINLP problems, however, are much more difficult to solve than MILP problems for which Branch and Bound methods perform reasonably well.

A large number of process synthesis, design and control problems in chemical engineering can be modeled as mixed integer nonlinear programming problems (Grossmann and Sargent, 1979; Kocis and Grossmann, 1987, 1988, 1989; Floudas et al., 1989; Salcedo, 1992; Ciric and Gu, 1994 etc.). They involve continuous (floating point) and integer variables. A common feature of this class of mathematical problem is the potential existence of nonconvexities due to the particular form of the objective function and/or the set of constraints. Due to their combinatorial nature, these problems are considered to be difficult.

The optimization of mixed integer nonlinear programming problems constitutes an active area of research. So far various methods such as branch and bound technique (Grossmann and Sargent, 1979), outer-approximation (OA)/equality-relaxation algorithm (Kocis & Grossmann, 1987, 1988, 1989), variant of OA method (Diwekar et al., 1992), adaptive random-search method (MSGA by Salcedo, 1992), branch-and-reduce algorithm (Ryoo and Sahinidis, 1995), MINLP Simplex Simulated Annealing Algorithm (M-SIMPASA by Cardoso et al., 1997), and genetic algorithm & evolution strategies (Costa and Olivera, 2001) have been used for solving nonconvex MINLP problems.

Gradient optimization techniques have only been able to tackle special formulations, where continuity or convexity had to be imposed, or by exploiting special mathematical structures. Stochastic algorithms, also known as adaptive random search methods, have tackled MINLP problems, mostly in the area of chemical engineering (Salcedo, 1992). These require neither the prior step of

identification or elimination of the sources of nonconvexities nor decomposition of the problem into sub problems which have to be iteratively solved. However, various problem-independent heuristics related to search interval compression and expansion and to shifting strategies are required for their effectiveness (Salcedo, 1992). Also, for large scale or very ill conditioned and highly constrained functions, these methods require the application of successive relaxations which may substantially increase the effort in identifying feasible regions and attaining the global optimum and hence suited for small to medium scale problems. Cardoso et al., (1997) compared the performance of the M-SIMPISA with MSGA (Slacedo, 1992). They concluded that for small-scale problems and with penalizing scheme its performance is comparable to MSGA algorithm, however, for large scale and /or ill conditioned problems, the M-SIMPISA algorithm performed better. Costa and Olivera (2001) studied seven test problems using GA & Evolution Strategies (ESs) and compared the results with M-SIMPISA algorithm. They found that the performance of M-SIMPISA with penalty is better than that of M-SIMPISA without penalty. Also the performance of M-SIMPISA is comparable to GA. Evolution Strategies emerged as the best algorithms in most of the problems studied. However, ESs exhibited difficulties in highly constrained problems but in general, they are found most efficient in terms of function evaluations. Also, all the algorithms (GA, ESs, and M-SIMPISA) are found to have great difficulties with multi-product batch plant problem (Grossmann and Sargent, 1979), which is highly constrained; the global optimum corresponds to a point where a very small variation in any of the continuous variables produces infeasibility.

In the present study (Babu and Angira, 2002b; Angira and Babu, 2005c), seven test problems on process synthesis & design are solved using DE and MDE algorithms. These problems are difficult non-convex optimization problems with

continuous and discrete variables. Also the two approaches for handling binary (discrete) variables are discussed and compared. The performance of MDE is compared with that of DE, GA, ESs, and M-SIMPSA algorithms.

### 4.3. Problem Formulation

The chemical process synthesis problem involves selecting the optimal flowsheet structure as well as the parameters which describe the operation of a desired process. In order to define the search space of candidate flowsheet alternatives, a superstructure is to be postulated based on preliminary screening. This superstructure can then be modeled as a MINLP problem of the form:

$$F = \min_{x,y} \mathbf{c}^T \mathbf{y} + f(x),$$

$$\text{Subject to. } h(x) = 0,$$

$$g(x) \leq 0,$$

$$\mathbf{Ax} = \mathbf{a},$$

$$\mathbf{By} + \mathbf{Cx} \leq \mathbf{d},$$

$$x \in X = \{x | x \in R^n, x^L \leq x \leq x^U\},$$

$$y \in Y = \{y | y \in \{0, 1\}^m, \mathbf{Ey} \leq \mathbf{e}\}.$$

where  $x$  is the vector of continuous variables specified in the compact set  $X$ , and  $y$  is the vector of 0-1 variables which must satisfy linear integer constraints  $\mathbf{Ey} \leq \mathbf{e}$ .  $f(x)$ ,  $h(x) = 0$ , and  $g(x) \leq 0$  represent nonlinear functions involved in the objective function, equations, and inequalities, respectively. Finally,  $\mathbf{Ax} = \mathbf{a}$  represents the subset of linear equations, while  $\mathbf{By} + \mathbf{Cx} \leq \mathbf{d}$  linear equalities or inequalities that involve the continuous and binary variables.

In the context of the synthesis problem, the continuous variables  $x$  include flows, pressures, temperatures, and sizes while the binary variables  $y$  denote the potential existence of process units which are embedded in the superstructure. The equations

$h(x) = 0$  and  $Ax = a$ , correspond to material and energy balances and design equations. Process specifications are represented by  $g(x) \leq 0$  and by lower and upper bounds on the variables in  $x$ . Logical constraints that must hold for a flow sheet configuration to be selected from within the superstructure are represented by  $By + Cx \leq d$  and  $Ey \leq e$ . The cost function involves fixed cost charges in the term  $c^T y$  for the investment, while revenues, operating costs, and size-dependent costs for the investment are included in the function  $f(x)$ .

## 4.4. Handling of Integer and Binary variables

### 4.4.1. Integer Variables

Original DE algorithm is only capable of handling continuous variables. Extending it to optimize integer variables, however, is very easy and requires only couple of simple modifications (Corne et al., 1999). First, integer values should be used to evaluate the objective function, even though DE itself still works internally with continuous floating- point values. Therefore:

$$y_i = \begin{cases} x_i & \text{for continuous variables} \\ INT(x_i) & \text{for integer variables} \end{cases}$$

$INT()$  is a function for converting a real value to an integer value by truncation. Additionally, truncation is performed here for evaluating trial vectors and for handling boundary constraints. Truncated values are not assigned elsewhere. Hence, DE works with a population of continuous variables regardless of the corresponding object variable type which is also essential for maintaining diversity of the population and the robustness of the algorithm.

Corne et al., (1999) also described the procedure to handle discrete variable as integer variables. Here, instead of discrete value  $x_i$  itself ( $i = 1 \dots n$ ), we may assign its

index  $i$  to  $x$ . Now the discrete variable can be handled as an integer variable that is boundary constrained to range 1, ...,  $n$ . To evaluate the objective function, the discrete value  $x_i$  is used instead of index  $i$ .

#### 4.4.2. Binary or discrete variables

In the present study, integer variables are handled in the same way as described above however, two different procedures are used to handle binary (discrete) variables. The first procedure is called *Approach-1* and second procedure is called *Approach-2* in the present study. These are described below:

##### 4.4.2.1. Approach-1

In this procedure, for the function evaluation, binary variables are handled as follows:

$$y_i = \begin{cases} 0 & \text{if } x_i \leq 0.5 \\ 1 & \text{otherwise} \end{cases}$$

Where  $x_i$  is a continuous variable  $0 \leq x_i \leq 1$ . However boundary constraint is handled in the same way as for continuous variables. The only difference is that lower and upper bound are set to zero and one respectively.

##### 4.4.2.2. Approach-2

It is a nonlinear transformation for modeling binary variables as continuous variables proposed by Li (1992). A binary variable  $y \in \{0, 1\}$  can be modeled as a continuous variable  $x \in [0, 1]$ , simply by the addition of the following constraint in the problem:

$$x(1-x) = 0, \quad 0 \leq x \leq 1$$

which forces  $x$  to take either 0 or 1. Hence with this transformation any MINLP model can be converted into an equivalent NLP model. The function  $x(1-x)$  is a non-convex nonlinear function. Li (1992) referred to this as the "binary condition" to

model binary variables and found this procedure to be more convenient than current approaches as branch-and-bound method and implicit enumeration method. The resulting NLP was solved using a modified penalty function method, however only local optimal solutions were found due to the addition of non-convexities. Also, the use of standard NLP solver like SQP (sequential quadratic programming) and GRG (generalized reduced gradient) is ruled out as they could be sensitive to initial guess and hence get stuck at local optima. Ryoo and Sahinidis (1995) proposed a specialized branch and reduce algorithm for the global optimization of NLPs and MINLPs wherein they refer to the usage of such procedure for handling binary and discrete variables. Munawar and Gudi (2004) proposed hybrid-DE (a combination of DE and GAMS) and used this binary condition to solve the benchmark test problems. The hybrid-DE exhibited relatively superior rates of convergence to the global optimum for the problems studied however the guarantee of global optimality still remained an issue. The present study evaluates the application of DE and MDE algorithms for solving the resulting nonconvex NLP problems to global optimality.

#### **4.5. Test Problems on Process Synthesis and Design**

To illustrate the applicability and efficiency of MDE and DE to the nonconvex MINLP problems, seven test problems on process synthesis and design proposed by different authors have been solved. These problems arise from the area of chemical engineering, and represent difficult nonconvex optimization problems, with continuous and discrete variables. Problem-2 and Problem-4, with equality constraints are reformulated (as Problem-2\* and Problem-4\*) by eliminating the equality constraints and incorporating them in inequality constraints and/or in objective function thereby reducing the number of constraints and decision variables.

Comparisons are made with GA, ESs, and M-SIMPISA (a algorithm based on the combination of the nonlinear simplex method of Nelder & Mead and Simulated Annealing). Table-4.1 shows characteristics of the seven test problems considered in the present study.

**Table-4.1. Characteristics of the test problems**

Problem	No. of Variables				No. of Constraints	
	Real	Integer	Binary	Total	Equality	Inequality
1	1	---	1	2	-	2
2	2	--	1	3	1	1
2*	1	---	1	2	-	1
3	2	---	1	3	-	3
4	7	---	2	9	6	4
4*	4	---	1	5	-	6
5	3	---	4	7	-	9
6	3	2	---	5	-	3
7	7	3	---	10	-	18

**Problem-1:** This example has a non-linear constraint and has been proposed by Kocis & Grossmann (1988). It has also been solved by other authors (Floudas et al., 1989; Ryoo and Sahinidis, 1995; Cardoso et al., 1997; and Costa and Olivera, 2001).

$$\text{Min } f(x, y) = 2x + y$$

Subject to

$$1.25 - x^2 - y \leq 0$$

$$x + y \leq 1.6$$

$$0 \leq x \leq 1.6$$

$$y \in \{0, 1\}$$

The global optimum is  $(x, y, f) = (0.5, 1; 2)$ .

**Problem-2:** This problem, with a non-linear constraint is proposed by Kocis & Grossmann (1988) and is also studied by Salcedo (1992), Cardoso et al., (1997), Costa and Olivera (2001).

$$\text{Min } f(x_1, x_2, y) = -y + 2x_1 + x_2$$

Subject to

$$x_1 - 2\exp(-x_2) = 0$$

$$-x_1 + x_2 + y \leq 0$$

$$0.5 \leq x_1 \leq 1.4$$

$$y \in \{0, 1\}$$

The global optimum is  $(x_1, x_2, y; f) = (1.375, 0.375, 1; 2.124)$ .

**Problem-2\***: Problem-2 can be reformulated, by eliminating the nonlinear equality constraint, as follows:

$$\text{Min } f(x_1, y) = -y + 2x_1 - \ln(x_1/2)$$

Subject to

$$-x_1 - \ln(x_1/2) + y \leq 0$$

$$0.5 \leq x_1 \leq 1.4$$

$$y \in \{0, 1\}$$

The global optimum is same as in Problem-2.

**Problem-3**: This problem was first studied by Floudas (1995) and is nonconvex because of the first constraint. It has also been solved by Cardoso et al., (1997), and Costa and Oliveira (2001).

$$\text{Min } f(x_1, x_2, y) = -0.7y + 5(x_1 - 0.5)^2 + 0.8$$

Subject to

$$-\exp(x_1 - 0.2) - x_2 \leq 0$$

$$x_2 + 1.1y \leq -1.0$$

$$x_1 - y \leq 0.2$$

$$0.2 \leq x_1 \leq 1$$

$$-2.22554 \leq x_2 \leq -1$$



$$y \in \{0, 1\}$$

The global optimum is  $(x_1, x_2, y; f) = (0.94194, -2.1, 1; 1.07654)$ .

**Problem-4:** It has been taken from Kocis & Grossmann (1989). The objective here is to select one between two candidate reactors (as shown in Fig. 4.1) in order to minimize the production cost. Also, it has been solved by Diwekar et al., (1992), Diwekar and Rubin (1993), Cardoso et al., (2001), and Costa and Oliveira (2001).

$$\text{Min } f(x, y_1, y_2, v_1, v_2) = 7.5y_1 + 5.5y_2 + 7v_1 + 6v_2 + 5x$$

Subject to

$$y_1 + y_2 = 1$$

$$z_1 = 0.9[1 - \exp(-0.5v_1)] x_1$$

$$z_2 = 0.8[1 - \exp(-0.4v_2)] x_2$$

$$z_1 + z_2 = 10$$

$$x_1 + x_2 = x$$

$$z_1y_1 + z_2y_2 = 10$$

$$v_1 \leq 10y_1$$

$$v_2 \leq 10y_2$$

$$x_1 \leq 20y_1$$

$$x_2 \leq 20y_2$$

$$x_1, x_2, z_1, z_2, v_1, v_2 \geq 0$$

$$y_1, y_2 \in \{0, 1\}$$

The binary variables  $y_1$  and  $y_2$  denote the existence (nonexistence) of reactor 1 and 2 when their value is 1 (0). In the objective function, there are fixed charges for purchasing reactor 1 (7.5) or reactor 2 (5.5), linear terms in  $v_1$  and  $v_2$  (reactor volumes) and the purchase price for raw material  $x$ . The two nonlinear equations are the input-output relations for the reactors which define the output flows ( $z_1$  and  $z_2$ ) in

terms of the input flows ( $x_1$  and  $x_2$ ) and the volumes. The raw material  $x$  is split into the reactor input flows  $x_1$  and  $x_2$ ; a total demand of 10 units must be met by the output flows  $z_1, z_2$ . The next four inequalities are logical constraints which insure that if a given reactor does not exist (e.g.  $y_1 = 0$ ), then the corresponding volume and feed stream are zero. The last constraint requires that either reactor 1 or 2 be selected. The suboptimal solution corresponding to  $(y_1, y_2) = (0, 1)$  has an objective function value of 107.376 at  $(x_1, x_2) = (0.0, 15.0)$  and  $(v_1, v_2) = (0.0, 4.479)$ .

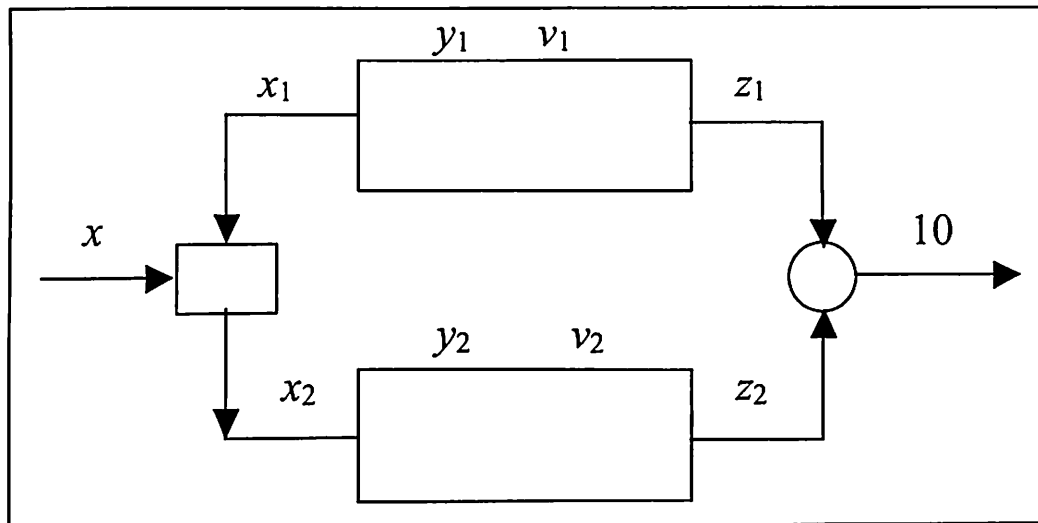


Fig. 4.1. Superstructure for two-reactor problem

The global optimum is:  $(x, y_1, y_2, v_1, v_2; f) = (13.36227, 1, 0, 3.514237, 0; 99.245209)$ .

**Problem-4\***: This can be reformulated without equality constraints as follows:

$$\text{Min } f(y_1, v_1, v_2) = 7.5y_1 + 5.5(1 - y_1) + 7v_1 + 6v_2 + 50 \frac{1 - y_1}{0.8[1 - \exp(-0.4v_2)]} + 50 \frac{y_1}{0.9[1 - \exp(-0.5v_1)]}$$

Subject to

$$0.9[1 - \exp(-0.5v_1)] - 2v_1 \leq 0$$

$$0.8[1 - \exp(-0.4v_2)] - 2(1 - y_1) \leq 0$$

$$v_1 \leq 10y_1$$

$$v_2 \leq 10(1 - y_1)$$

$$v_1, v_2 \geq 0$$

$$y_1 \in \{0, 1\}$$

The global optimum is same as in Problem-4.

**Problem-5:** This problem was studied by Floudas et al. (1989), Salcedo (1992), Ryoo and Sahinidis (1995), Cardoso et al. (1997), and Costa and Oliveira (2001). This problem features nonlinearities in both continuous and binary variables and has seven degrees of freedom.

$$\text{Min } f(x_1, x_2, x_3, y_1, y_2, y_3, y_4) = (y_1 - 1)^2 + (y_2 - 1)^2 + (y_3 - 1)^2 - \ln(y_4 + 1) + (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2.$$

Subject to

$$y_1 + y_2 + y_3 + x_1 + x_2 + x_3 \leq 5$$

$$y_3^2 + x_1^2 + x_2^2 + x_3^2 \leq 5.5$$

$$y_1 + x_1 \leq 1.2$$

$$y_2 + x_2 \leq 1.8$$

$$y_3 + x_3 \leq 2.5$$

$$y_4 + x_1 \leq 1.2$$

$$y_2^2 + x_2^2 \leq 1.64$$

$$y_3^2 + x_3^2 \leq 4.25$$

$$y_2^2 + x_3^2 \leq 4.64$$

$$x_1, x_2, x_3 \geq 0$$

$$y_1, y_2, y_3, y_4 \in \{0, 1\}$$

The global optimum is  $(x_1, x_2, x_3, y_1, y_2, y_3, y_4; f) = (0.2, 1.28062, 1.95448, 1, 0, 0, 1; 3.557473)$ .

**Problem-6:** It is a maximization problem taken from Wong (1990) and is also studied by Cardoso et al. (1997), and Costa and Oliveira (2001).

$$\text{Max } f(x_1, x_2, x_3, y_1, y_2) = -5.357854x_1^2 - 0.835689y_1x_3 - 37.29329y_1 + 40792.141.$$

Subject to

$$a_1 + a_2y_2x_3 + a_3y_1x_2 - a_4x_1x_3 \leq 92$$

$$a_5 + a_6y_2x_3 + a_7y_1y_2 + a_8x_1^2 - 90 \leq 20$$

$$a_9 + a_{10}x_1x_3 + a_{11}y_1x_1 + a_{12}x_1x_2 - 20 \leq 5$$

$$27 \leq x_1, x_2, x_3 \leq 45$$

$$y_1 \in \{78, \dots, 102\}, \text{ integer}$$

$$y_2 \in \{33, \dots, 45\}, \text{ integer}$$

where  $a_1$  to  $a_{12}$  are constants the values of which are given in Table-4.2. The global optimum (for any combination of  $x_2, y_2$ ) is:  $(x_1, x_3, y_1; f) = (27, 27, 78; 32217.4)$ .

**Table-4.2. Constants for Problem-6**

$a_1 = 85.334407$	$a_5 = 80.51249$	$a_9 = 9.300961$
$a_2 = 0.0056858$	$a_6 = 0.0071317$	$a_{10} = 0.0047026$
$a_3 = 0.0006262$	$a_7 = 0.0029955$	$a_{11} = 0.0012547$
$a_4 = 0.0022053$	$a_8 = 0.0021813$	$a_{12} = 0.0019085$

**Problem-7:** This is a multi-product batch plant problem with  $M$  serial processing stages, where fixed amounts  $Q_i$  from  $N$  products must be produced. Many researchers (Grossmann and Sargent (1979), Kocis and Grossmann (1988), Salcedo (1992), Cardoso (1997), and Costa and Oliviera (2001)) studied this problem.

$$\text{Min } f = \sum_{j=1}^M \alpha N_j V_j^\beta$$

Subject to

$$\sum_{i=1}^N \frac{Q_i T_{Li}}{B_i} \leq H$$

$$V_j \geq S_{ij}B_i$$

$$N_j T_{Li} \geq t_{ij}$$

$$1 \leq N_j \leq N_j^u$$

$$V_j^l \leq V_j \leq V_j^u$$

$$T_{Li}^l \leq T_{Li} \leq T_{Li}^u$$

$$B_j^l \leq B_j \leq B_j^u$$

where, for the specific problem considered,  $M = 3$ ,  $N = 2$ ,  $H = 6000$ ,  $\alpha_j = 250$ ,  $\beta_j = 0.6$ ,  $N_j^u = 3$ ,  $V_j^l = 250$  and  $V_j^u = 2500$ . The values of  $T_{Li}^l$ ,  $T_{Li}^u$ ,  $B_j^l$  and  $B_j^u$  are given by:

$$T_{Li}^l = \max t_{ij}/N_j^u$$

$$T_{Li}^u = \max t_{ij}$$

$$B_j^l = Q_i * T_{Li}/H$$

$$B_j^u = \min (Q_i, \min_j V_j^u/S_{ij})$$

The values of  $S_{ij}$  and  $t_{ij}$  [ $i = 1$  to 2 (rows); and  $j = 1$  to 3 (columns)] are given in Table-4.3. The global optimum is:  $(N_1, N_2, N_3, V_1, V_2, V_3, B_1, B_2, T_1, T_2; f) = (1, 1, 1, 480, 720, 960, 240, 120, 20, 16; 38499.8)$ .

**Table-4.3. Values of  $S_{ij}$  and  $t_{ij}$  of Problem-7**

$S_{ij}$			$t_{ij}$		
2	3	4	8	20	8
4	6	3	16	4	4

## 4.6. Results and Discussion

For each problem ten experiments are carried out with different seed values. *NFE*, *NRC* and CPU-time in the subsequent tables are the mean values of the ten experiments. The stopping criteria adopted for MDE and DE is to terminate the search process when one of the following conditions is satisfied: (1) the maximum number

of generations is reached (assumed 10,000 generations for Problem-7 & 5000 generations for other problems). (2)  $|f_{\max}^k - f_{\min}^k| < 10^{-5}$  where  $f$  is the value of objective function for  $k$ -th generation.

#### 4.6.1. Approach-1

Table-4.4 and Table-4.5 show the results obtained using DE and MDE (with/without forcing the bound on variables) respectively using Approach-1. From Table-4.4, it is clear that in all the seven problems, *NFE* for forced bound method is less than that for method without forcing the bound but the difference is quite significant for problems 3, 4, 5, 6, and 7. *NFE* for method without forcing the bound is about 3.4 times for Problem-3, 2.03 times for Problem-4\*, 1.12 times for Problem-5, 6.15 times for Problem-6, and 1.94 times for Problem-7, more than that of forced bound method.

**Table-4.4. Results of DE using Approach-1**

Problem No.	NFE/NRC/CPU-time		Key parameters (NP/CR/F)
	FBM	MWFB	
1	802/90/0.022	812/100/0.0	20/0.8/0.5
2*	610/100/0.011	626/100/0.011	20/0.8/0.5
3	801/10/0.011	2727/100/0.071	30/0.8/0.7
4*	1080/100/0.040	2196/100/0.083	30/0.8/0.5
5	11739/100/0.610	13265/90/0.696	30/0.9/0.6
6	1045/100/0.039	6430/100/0.280	20/0.8/0.5
7	46090/100/3.225	89490/0**/6.291	100/0.8/0.5

\*\* Converged to a non-optimal solution

It is important to note that *NRC* is not 100% for all the problems using either forced bound method or method without forcing the bound. Excepting for Problem-7, the *NRC* is almost 100% using method without forcing the bound. Similarly, for forced bound method *NRC* is almost 100% excepting Problem-3. It is because when upper limit of bound is violated, the value of variable is forced to the upper limit that resulted in convergence to non-optimal solution. However, for problem-7, the *NRC* is 100% for forced bound method and 0.0% for method without forcing the bound, as

global optimum is located on the bound of decision variables. CPU-time is found to be more in case of method without forcing the bound.

It is observed from Table-4.5 that *NRC* is 100% for all the problems excepting Problem-7 using method without forcing the bound while *NRC* is zero and 80% for Problem-3 & 5 respectively using forced bound method. *NFE* and hence CPU-time is found to be more for method without forcing the bound as compared to forced bound method. In all the seven problems, *NFE* for forced bound is less than that for without forcing but the difference is quite significant for problems 3, 4\*, 5, 6, and 7 than for problems-1 & 2\*. *NFE* for without forcing is about 1.86 times for Problem-3, 1.52 times for Problem-4\*, 1.17 times for Problem-5, 6.35 times for Problem-6, and 1.96 times for Problem-7, more than that of forced bound method. This observation is similar to that seen for results reported in Table-4.4 using DE.

**Table-4.5. Results of MDE using Approach-1**

Problem No.	<i>NFE/NRC/CPU-time</i>		Key parameters ( <i>NP/CR/F</i> )
	FBM	MWFB	
1	690/100/0.000	705/100/0.000	20/0.8/0.5
2*	490/100/0.011	490/100/0.000	20/0.8/0.5
3	1062/0**/0.027	1974/100/0.055	30/0.8/0.5
4*	1179/80/0.033	1797/100/0.055	30/0.8/0.5
5	10129/100/0.527	11914/100/0.621	30/0.9/0.6
6	865/100/0.030	5495/100/0.220	20/0.8/0.5
7	40550/100/2.846	79380/0**/5.544	100/0.8/0.5

\*\* Converged to a non-optimal solution

#### 4.6.1.1. Comparison of DE and MDE

Fig. 4.2 and Fig. 4.3 show the comparison of DE and MDE in terms of *NFE* and *NRC* respectively using method without forcing the bound. As can be seen from Fig. 4.2. *NFE* using MDE is less as compared to DE in all the seven problems. *NFE* is about 13.17%, 21.73%, 27.61%, 18.17%, 10.18%, 14.54%, and 11.3% more, respectively for Problem-1, 2, 3, 4, 5, 6, and 7 in case of DE as compared to MDE. *NRC* for MDE, as shown in Fig. 4.3, is 100% in all the problems except Problem-7

(for which *NRC* is zero). For DE too, it is 100% for all the problems except Problem-5 (for which *NRC* is 90).

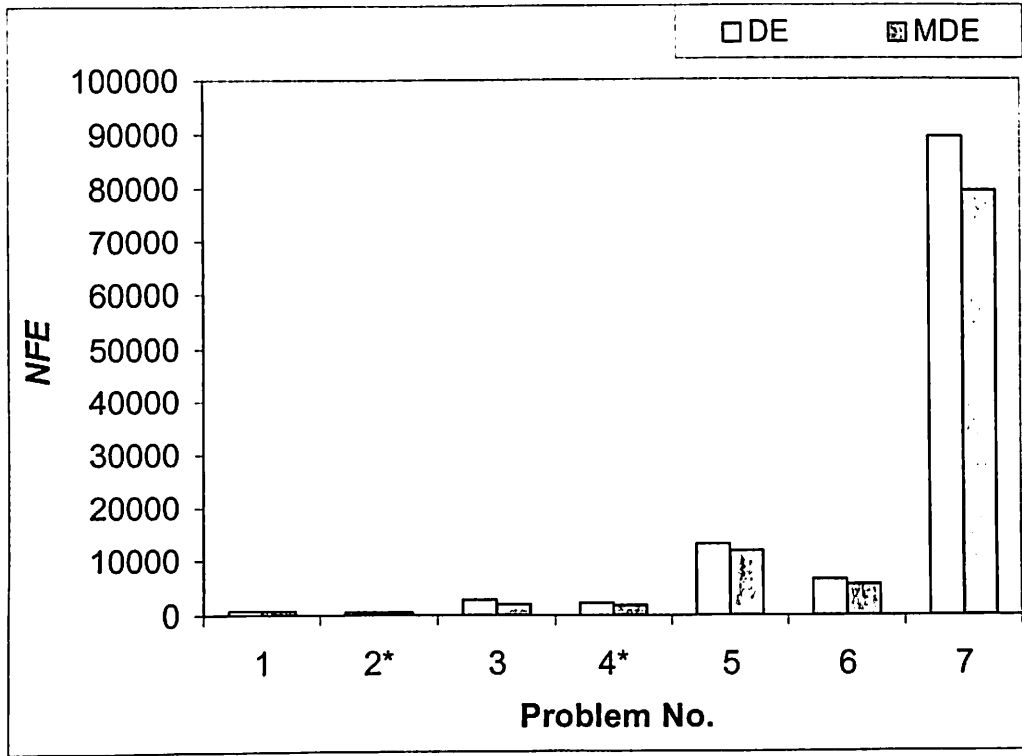


Fig. 4.2. *NFE* variation using DE and MDE (MWFB, Approach-1)

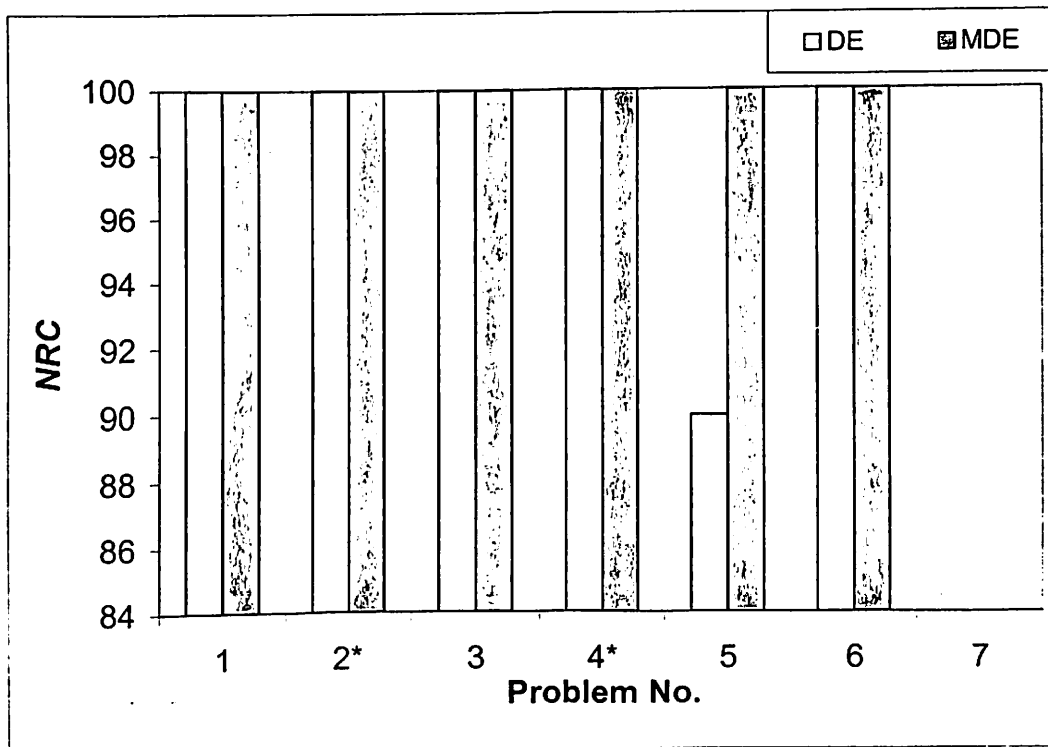


Fig. 4.3. *NRC* variation using DE and MDE (MWFB, Approach-1)



Fig. 4.4 and Fig. 4.5 show the comparison of DE and MDE in terms of *NFE* and *NRC* respectively using forced bound method. As can be seen from Fig. 4.4, *NFE* using MDE is less as compared to DE for Problem-1, 2\*, 5, 6, and 7. However, *NFE* using DE is less for Problem-3 & 4\*. *NFE* is about 13.97%, 19.67%, 13.71%, 17.22%, and 12.02% more, respectively for Problem-1, 2, 5, 6, and 7 in case of DE as compared to MDE. As shown in Fig. 4.5, *NRC* for MDE is 100% in all the problems except Problem-5 (for which *NRC* is 90). For DE too, it is 100% for all the problems except Problem-1 & 3 (for which *NRC* is 90 & 10 respectively).

#### 4.6.2. Approach-2

Table-4.6 and Table-4.7 show the results obtained using DE and MDE (with/without forcing the bound on variables) respectively using Approach-2. It is clear that in all the seven problems, *NFE* for forced bound method is significantly less than that for without forcing. *NFE* for without forcing is about 49.5%, 50.9%, 70.9%, 79.9%, 69.7%, and 53.07% respectively for Problem-1, 2\*, 3, 4\*, 5, and 7, more than that for forced bound method. It is important to note that *NRC* is not 100% for all the problems using either forced bound method or without forcing the method. It is important to note that for Problem-7, the *NRC* is 100% using forced bound method and 0.0% using without forcing method as found for Approach-1. CPU-time is found to be more in case of without forcing the bound method.

**Table-4.6. Results of DE using Approach-2**

Problem No.	<i>NFE/NRC/CPU-time</i>		Key parameters ( <i>NP/CR/F</i> )
	FBM	MWFB	
1	964/90/0.011	1910/60/0.055	20/0.8/0.5
2*	654/100/0.011	1332/100/0.011	20/0.8/0.5
3	801/10/0.027	2754/100/0.083	30/0.8/0.7
4*	1071/100/0.055	5328/100/0.187	30/0.8/0.5
5	16114/80/0.852	53158/90/3.016	30/0.9/0.6
7	57330/100/4.022	122170/0**/8.681	100/0.8/0.5

\*\* Converged to a non-optimal solution

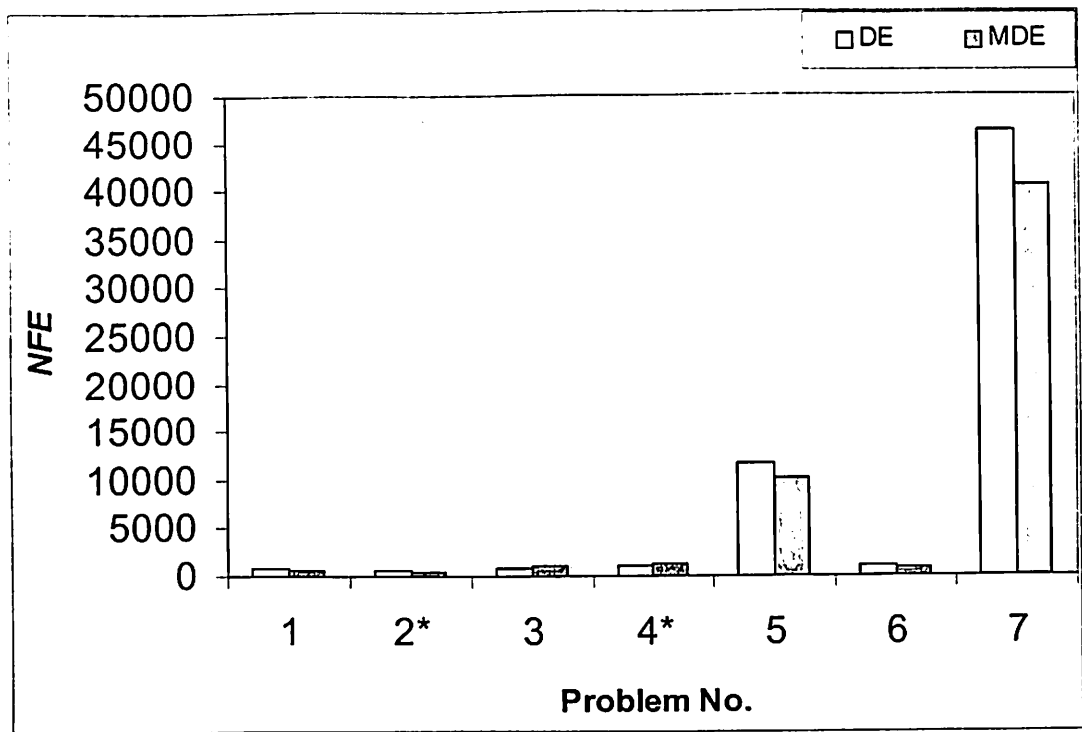


Fig. 4.4. *NFE* variation using DE and MDE (FBM, Approach-1)

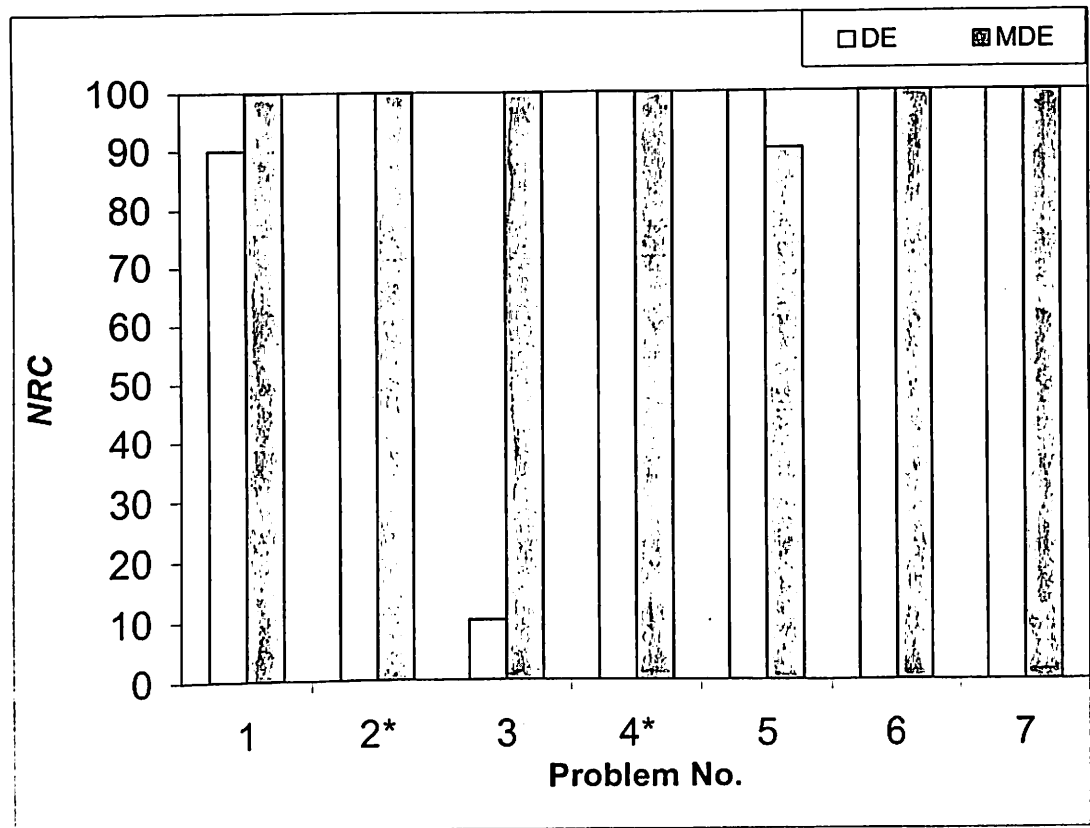


Fig. 4.5. *NRC* variation using DE and MDE (FBM, Approach-1)

**Table-4.7. Results of MDE using Approach-2**

Problem No.	NFE/NRC/CPU-time		Key parameters (NP/CR/F)
	FBM	MWFB	
1	742/60/0.011	1408/90/0.022	20/0.8/0.5
2*	560/100/0.016	1218/100/0.055	20/0.8/0.5
3	882/20/0.030	3297/100/0.110	30/0.8/0.5
4*	927/100/0.027	4602/100/0.176	30/0.8/0.5
5	13566/60/0.703	45801/90/2.511	30/0.9/0.6
7	52350/100/3.621	110970/0**/7.786	100/0.8/0.5

\*\* Converged to a non-optimal solution

It is observed from Table-4.7 that *NRC* is not 100% for all the problems using either forced bound method or method without forcing the bound but certainly it is better or equal in case of method without forcing the bound for all the problems excepting Problem-7 (where *NRC* is zero). *NFE* and hence CPU-time is found to be twice or more for without forcing the bound method as compared to forced bound method. To be precise, *NFE* for without forcing is about 47.3%, 54.0%, 73.25%, 79.9%, 70.4%, and 52.8% respectively for problems 1, 2\*, 3, 4\*, 5, and 7, more than that of forced bound method. This observation is similar to that seen for results reported in Table-4.6 using DE.

#### 4.6.2.1. Comparison of DE and MDE

Fig. 4.6 and Fig. 4.7 show the comparison of DE and MDE in terms of *NFE* and *NRC* respectively using method without forcing the bound and Approach-2. As can be seen from Fig. 4.6, *NFE* using MDE is less as compared to using DE for all the problems except Problem-3, where *NFE* using DE is less than that of MDE. *NFE* is about 26.28%, 8.56%, 13.62%, 13.84%, and 9.17% more, respectively for problem-1, 2\*, 4\*, 5, and 7 in case of DE as compared to MDE. As shown in Fig. 4.7, *NRC* for MDE and DE is same for all the problems except Problem-1 where *NRC* for DE is 60% as compared to 90% for MDE. It is to be noted that *NRC* is zero for Problem-7 using both DE and MDE.

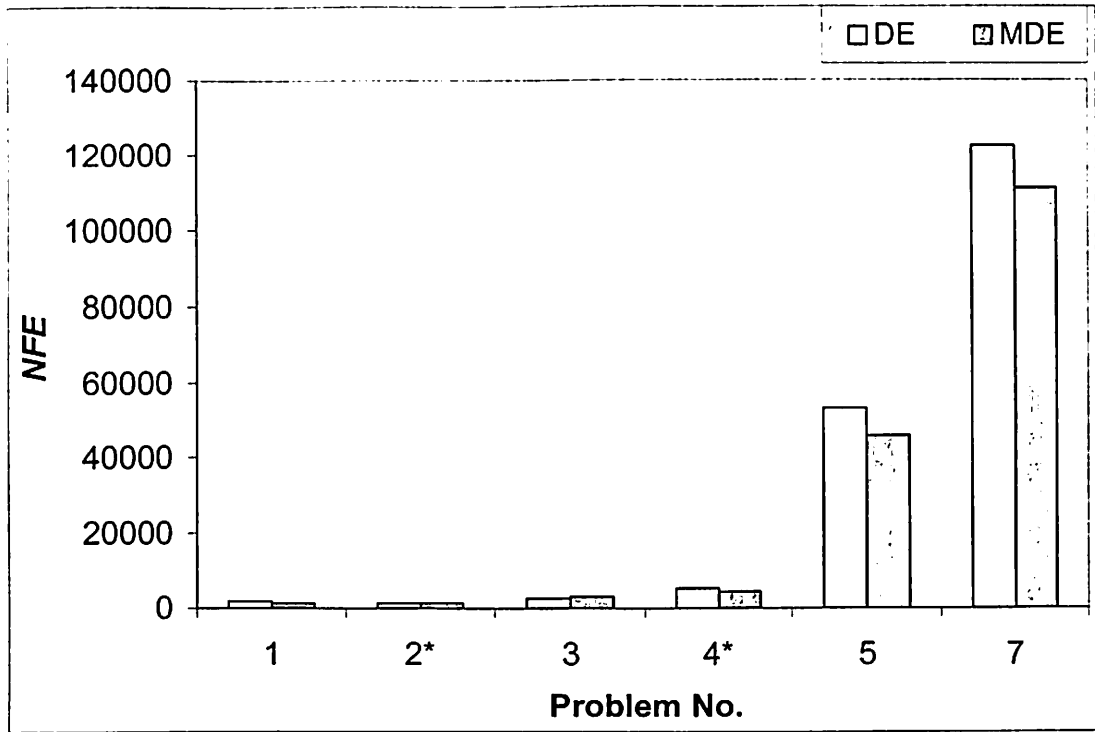


Fig. 4.6. *NFE* variation using DE and MDE (MWFB, Approach-2)

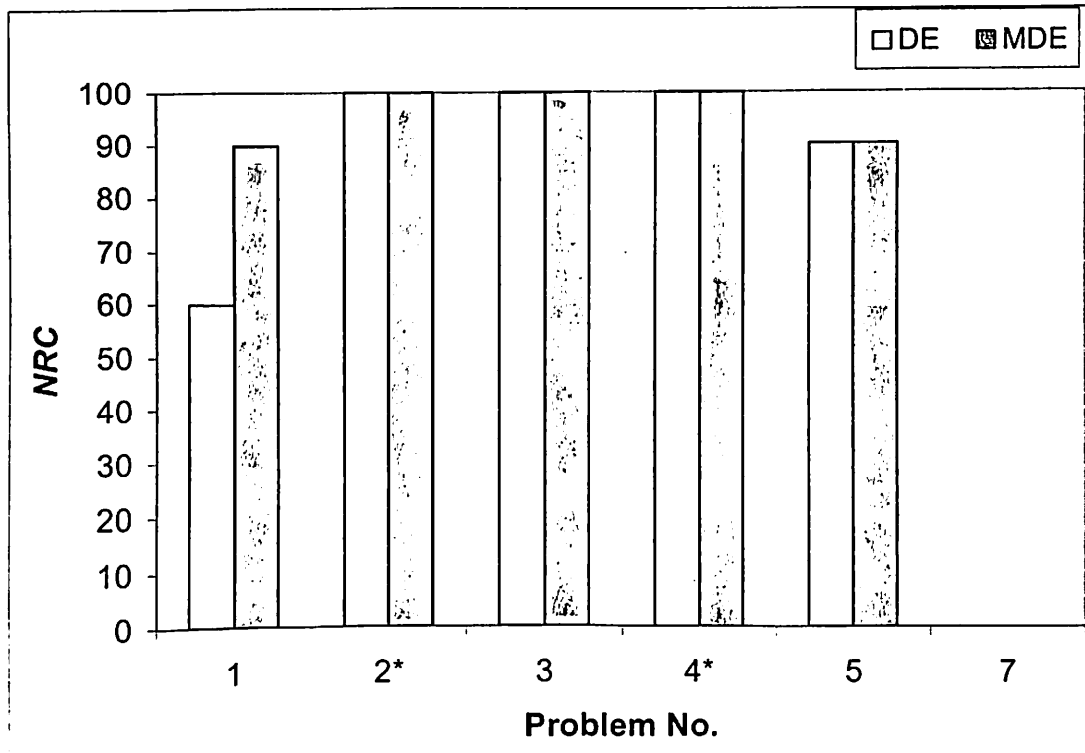


Fig. 4.7. *NRC* variation using DE and MDE (MWFB, Approach-2)

Fig. 4.8 and Fig. 4.9 show the comparison of DE and MDE in terms of *NFE* and *NRC* respectively using forced bound method and Approach-2. As can be seen from Fig. 4.8, *NFE* using MDE is less as compared to DE for all the problems except for Problem-3 where *NFE* using DE is slightly less (about 9.0%) than that of MDE. *NFE* is about 23.03%, 14.37%, 13.44%, 15.81%, and 8.68% more, respectively for Problem-1, 2\*, 4\*, 5, and 7 in case of DE as compared to MDE. As shown in Fig. 4.9, *NRC* for MDE and DE is 100% for Problem-2, 4\*, and 7. For Problem-1 & 5, *NRC* is higher for DE as compared to MDE but for Problem-3, *NRC* for DE is only 10% as compared to 20% for MDE. It is to be noted that *NRC* is not 100% for Problem-1, 3, and 5 using both DE and MDE. As compared to without forcing, the *NRC* is less for all the problems using forced bound method. The savings in *NFE* using MDE is almost same in both the methods (forced bound method and method without forcing the bound). This indicates that the method without forcing the bound is better than the forced bound method.

#### 4.6.3. Comparison of Approach-1 and Approach-2

Fig. 4.10 and Fig. 4.11 show the comparison of Approach-1 and Approach-2 for DE in terms of *NFE* and *NRC* respectively using method without forcing the bound. As can be seen from Fig. 4.10, *NFE* using Approach-2 is significantly more as compared to Approach-1 for all the problems except for Problem-3 where *NFE* using is almost same (2727 using Approach-1 and 2754 using Approach-2 respectively) in both the approaches. *NFE* is about 57.49%, 53.00%, 58.78%, 75.04%, and 26.75% more, respectively for Problem-1, 2\*, 4\*, 5, and 7 in case of Approach-2 as compared to Approach-1. This is because of addition of constraints (as binary or discrete variable is modeled as continuous variable) and addition of nonconvexities to the problem in case of Approach-2.

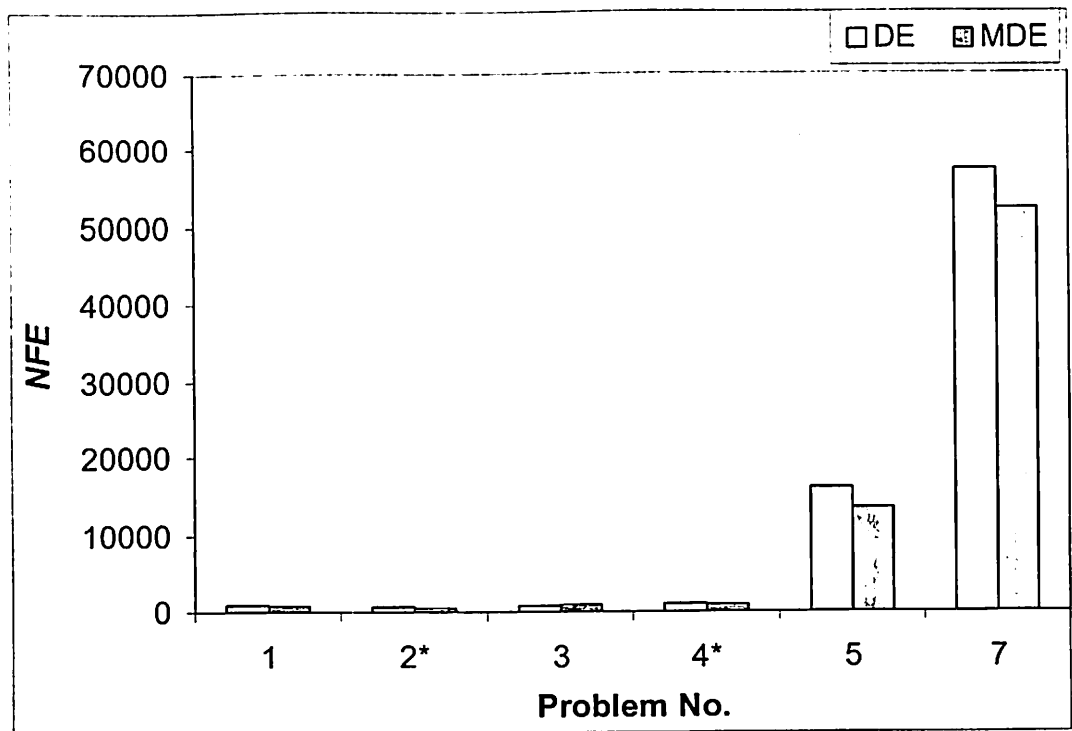


Fig. 4.8. *NFE* variation using DE and MDE (FBM, Approach-2)

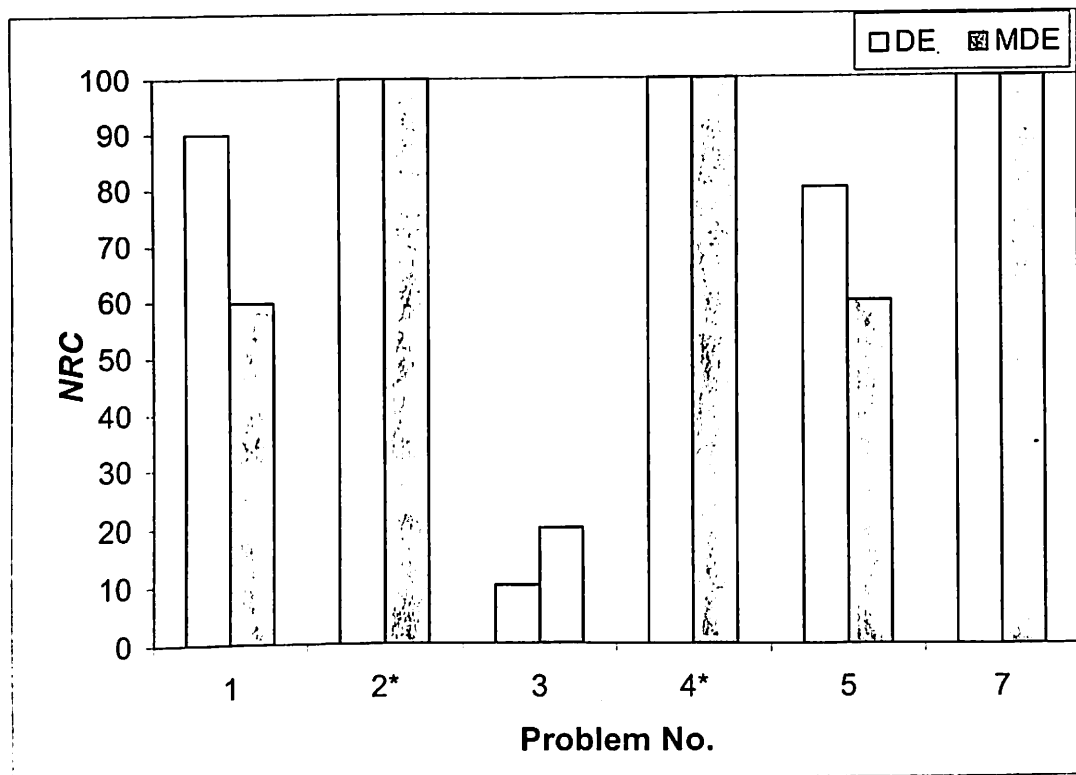


Fig. 4.9. *NRC* variation using DE and MDE (FBM, Approach-2)

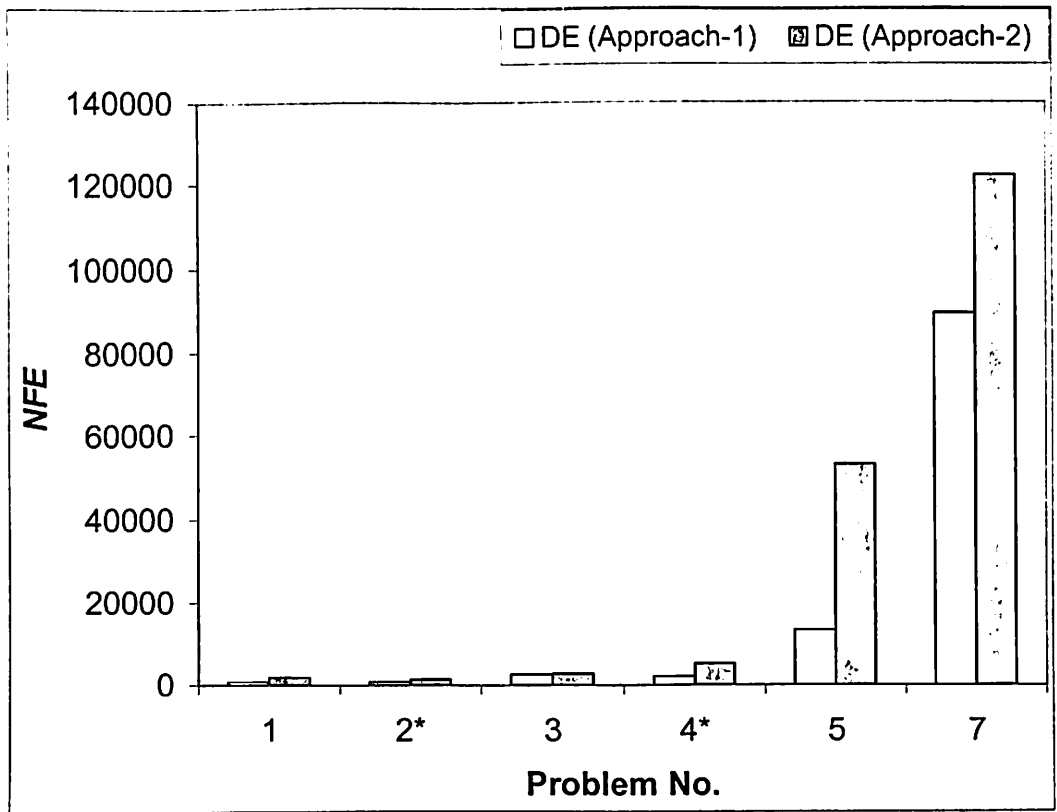


Fig. 4.10. *NFE* variation for DE using Approach-1 and Approach-2 (MWFB)

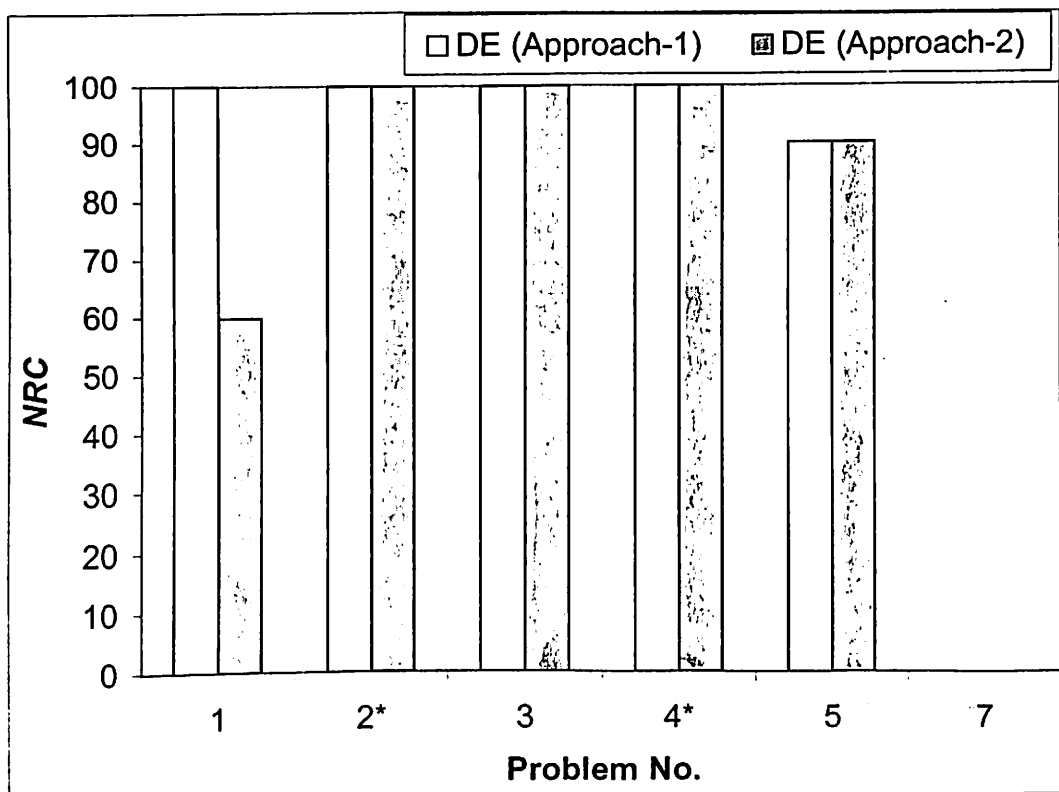


Fig. 4.11. *NRC* variation for DE using Approach-1 and Approach-2 (MWFB)

As shown in Fig. 4.11, *NRC* for Approach-1 and 2 is same i.e. 100%, 100%, 100%, 90%, and zero respectively for the Problems-2\*, 3, 4\*, 5, and 7. For Problem-1, *NRC* is 100% for Approach-1 as compared to 60% for Approach-2. It is to be noted that *NRC* is zero for Problem-7 using both the approaches. Therefore, using DE, the Approach-1 is found to be better than Approach-2 in terms of both *NFE* and *NRC*.

Fig. 4.12 and Fig. 4.13 show the comparison of Approach-1 and Approach-2 for MDE in terms of *NFE* and *NRC* respectively using method without forcing the bound. As can be seen from Fig. 4.12, *NFE* using Approach-2 is significantly more as compared to Approach-1 for all the problems. *NFE* is about 50.0%, 59.77%, 40.13%, 60.95%, 74%, and 28.47% more, respectively for problem-1, 2\*, 3, 4\*, 5, and 7 in case of Approach-2 as compared to Approach-1.

As shown in Fig. 4.13, *NRC* for Approach-1 and 2 is same i.e. 100%, 100%, 100%, and zero respectively for the problems-2\*, 3, 4\*, and 7. For Problem-1, *NRC* is 100% for Approach-1 as compared to 90% for Approach-2. It is to be noted that *NRC* is zero for Problem-7 using both approaches. Therefore, using MDE, the Approach-1 is found to be better than Approach-2 in terms of both *NFE* and *NRC*. Also, it is important to note that *NRC* using MDE is 100% for all the problems except for Problem-7. But using DE it is 90% for problem-5 and zero for Problem-7. Hence, MDE using Approach-1 seems to be a better strategy for solving such types of problems.

Fig. 4.14 and Fig. 4.15 show the comparison of Approach-1 and Approach-2 for DE in terms of *NFE* and *NRC* respectively using forced bound method. As can be seen from Fig. 4.14, *NFE* using Approach-2 is more as compared to Approach-1 for the Problem-1, 2, 5, and 7. For Problem-3 & 4\*, where *NFE* is almost same (801 & 1080 using Approach-1 and 801 & 1071 using Approach-2 respectively) in both the



approaches using DE. *NFE* is about 16.80%, 6.73%, 27.15%, and 19.61% more, respectively for Problem-1, 2\*, 5, and 7 in case of Approach-2 as compared to Approach-1.

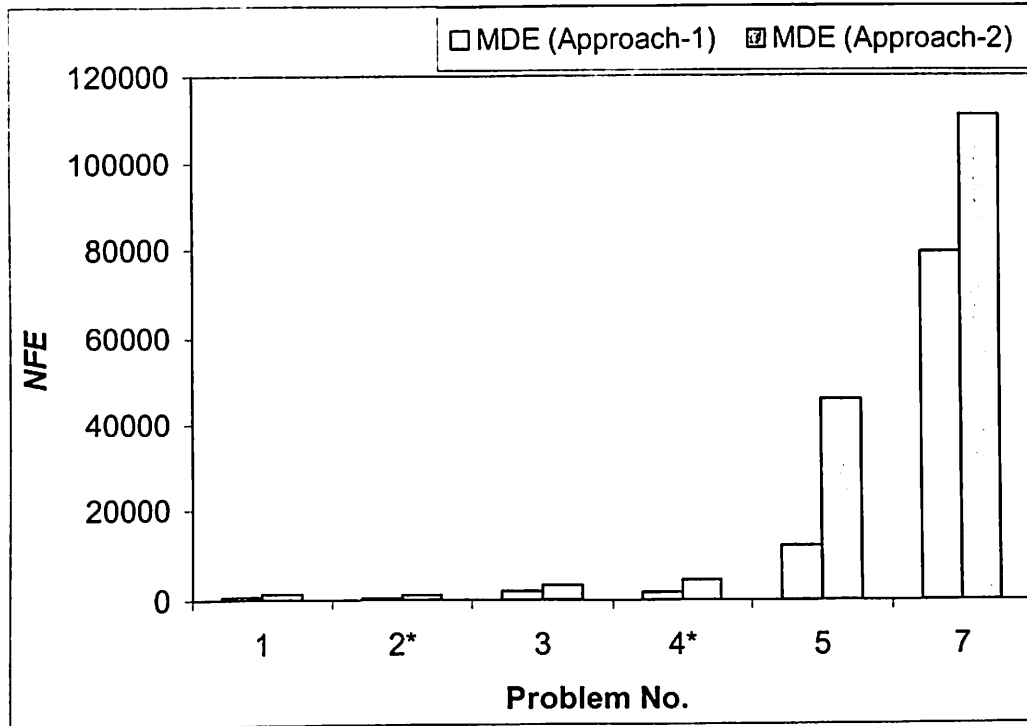


Fig. 4.12. *NFE* variation for MDE using Approach-1 and Approach-2 (MWFB)

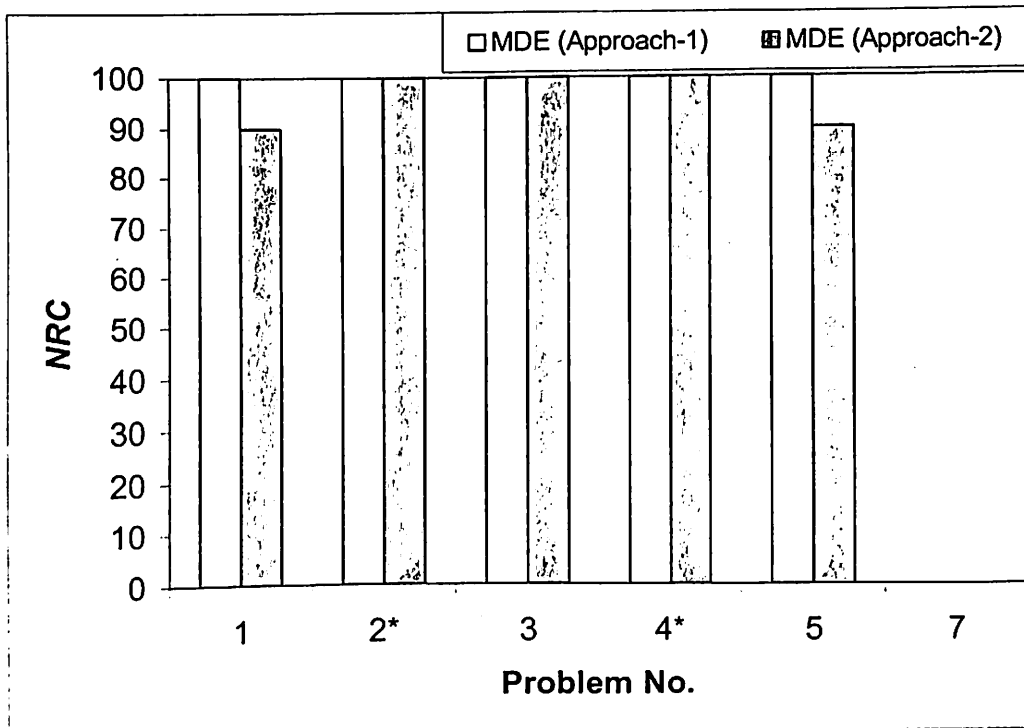


Fig. 4.13. *NRC* variation for MDE using Approach-1 and Approach-2 (MWFB)

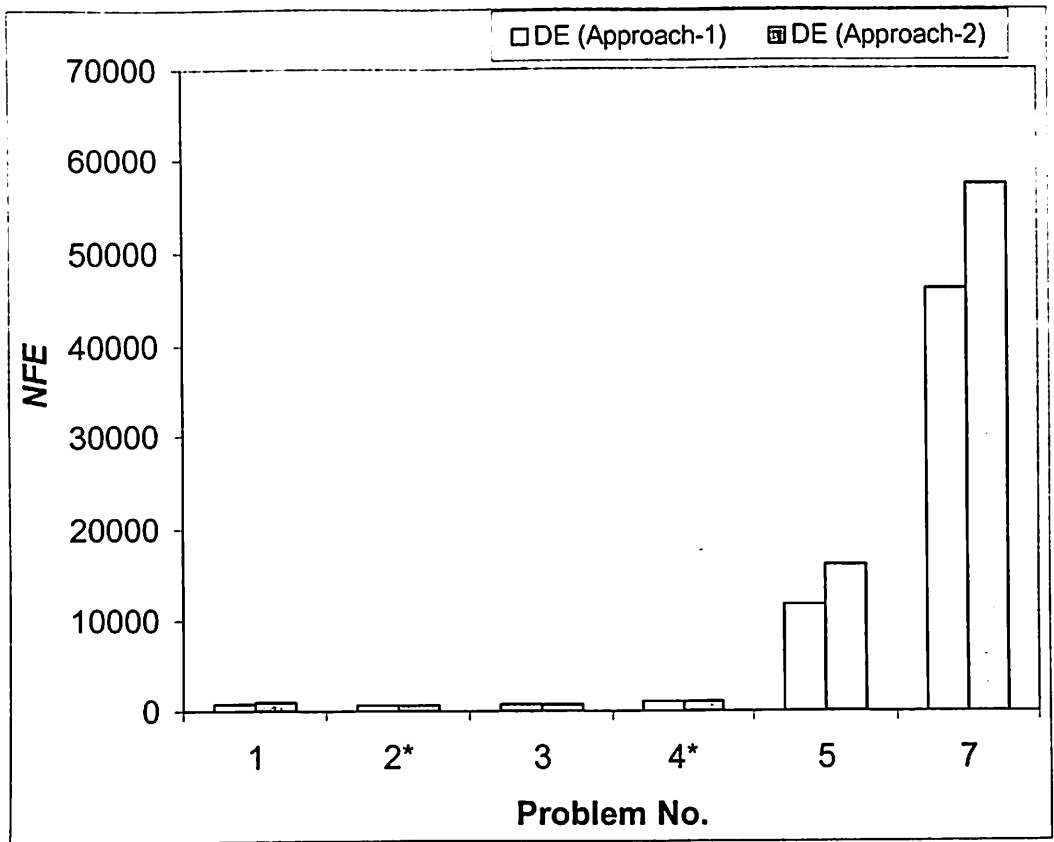


Fig. 4.14. *NFE* variation for DE using Approach-1 and Approach-2 (FBM)

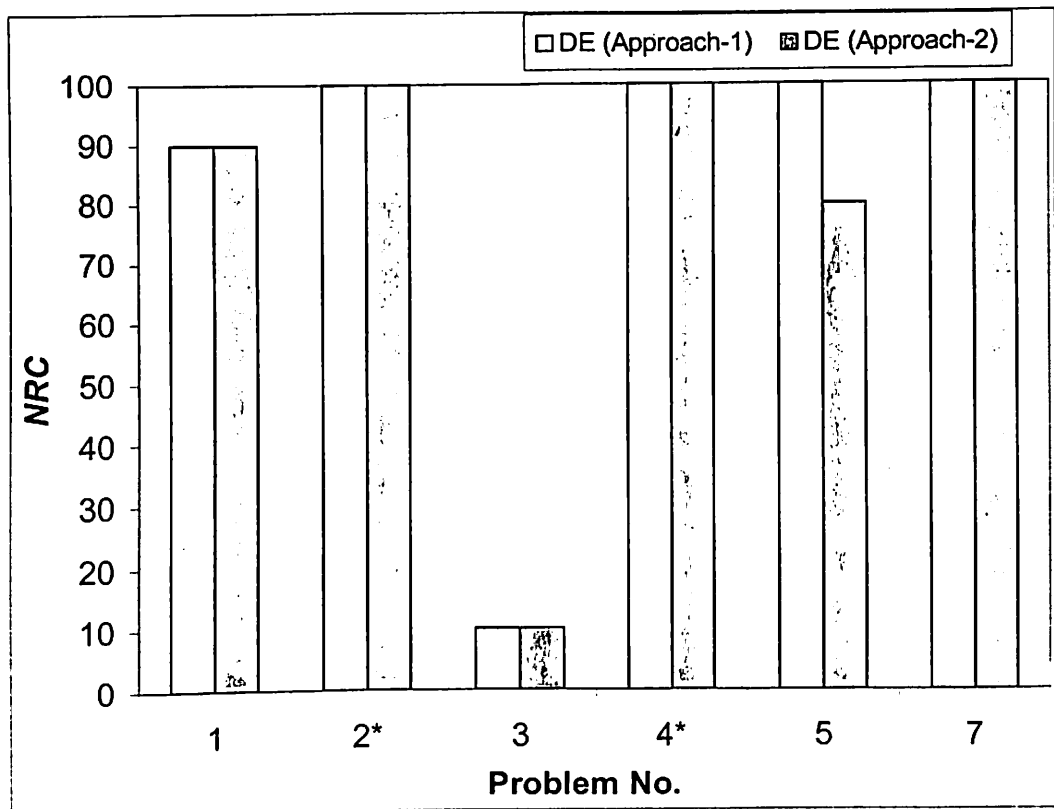


Fig. 4.15. *NRC* variation for DE using Approach-1 and Approach-2 (FBM)

As shown in Fig. 4.15, *NRC* for Approach-1 and 2 is same i.e. 90%, 100%, 10%, 100%, and 100% respectively for the Problems-1, 2\*, 3, 4\*, and 7. For Problem-5, *NRC* is 100% for Approach-1 as compared to 80% for Approach-2.

Fig. 4.16 and Fig. 4.17 show the comparison of Approach-1 and Approach-2 for MDE in terms of *NFE* and *NRC* respectively using forced bound method. As can be seen from Fig. 4.16, *NFE* using Approach-2 is more as compared to Approach-1 for all the problems except for Problem-3 & 4\* (where *NFE* is about 16.95% & 21.37% more using Approach-1 than that of Approach-2, respectively for Problem-3 & 4\*). *NFE* is about 7.0%, 12.50%, 25.33%, and 22.54% more, respectively for problem-1, 2\*, 5, and 7 in case of Approach-2 as compared to Approach-1.

As shown in Fig. 4.17, *NRC* for Approach-1 and 2 is same i.e. 100%, 100%, and 100%, respectively for the Problems-2\*, 4\*, and 7. For Problem-1, 3, and 5, *NRC* for Approach-1 is higher as compared to Approach-2. Therefore, using MDE, the Approach-1 is found to be better than Approach-2 in terms of both *NFE* and *NRC*. This indicates that nonconvexity (induced due to modeling of binary variable as continuous variable) affects the *NRC* (Fig. 4.11, 4.13, 4.15, and 4.17). Also, it is important to note that *NRC* using MDE is higher than that of using DE (Fig. 4.15 & Fig. 4.17). Therefore, MDE using Approach-1 seems to be a better strategy for solving such types of problems.

#### 4.6.4. Comparison of MDE, GA, M-SIMPISA, and ES ( $\mu + \lambda$ )

In the present study (Angira and Babu, 2005c), DE and MDE are used to solve process synthesis and design problems and their performance is compared. Further, the performance of MDE is evaluated and compared with that of GA, M-SIMPISA, and ES ( $\mu + \lambda$ ) algorithms. Table-4.8 shows the comparison of MDE with GA, M-

SIMPISA, M-SIMPISA-pen, and ES ( $\mu + \lambda$ ). The *NFE* in MDE is about 82% less than of that in GA for all the problems (The range is 82.0% to 98.16% to be precise).

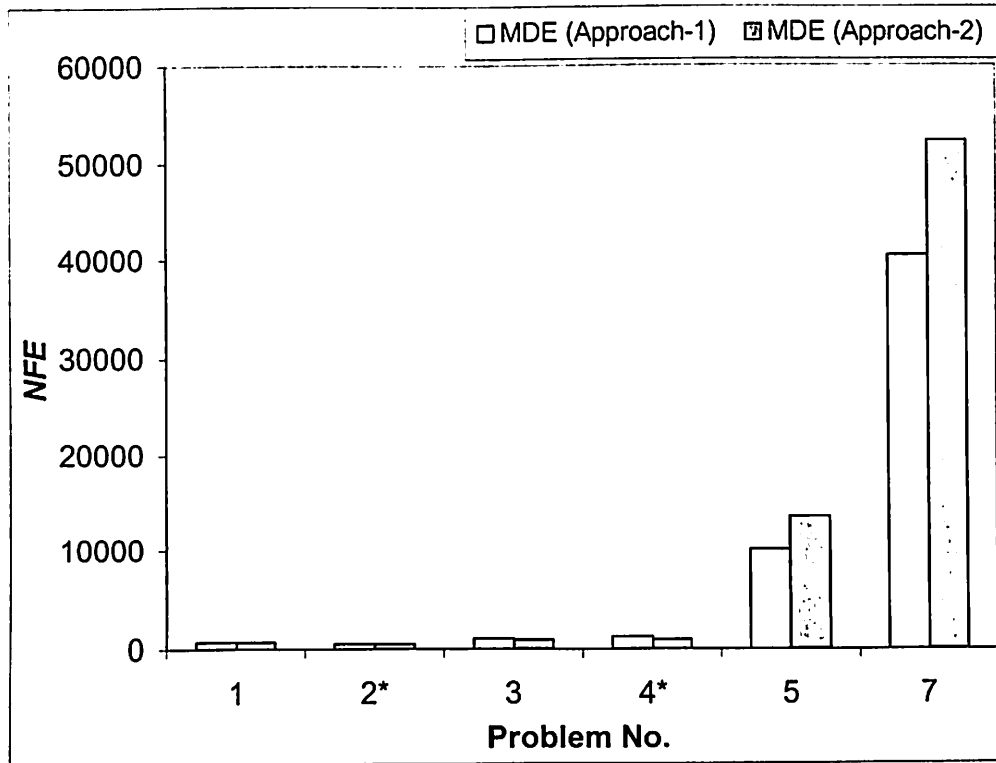


Fig. 4.16. *NFE* variation for MDE using Approach-1 and Approach-2 (FBM)

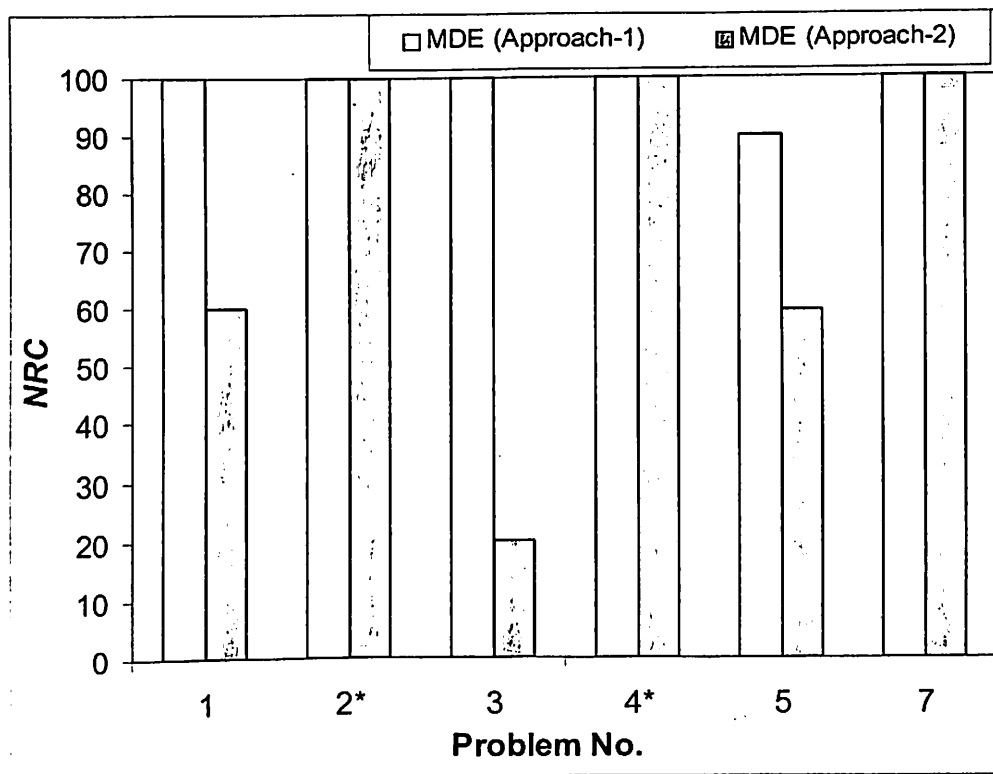


Fig. 4.17. *NRC* variation for MDE using Approach-1 and Approach-2 (FBM)

Similarly, the *NFE* in MDE is about 81% to 96.0% less than that of M-SIMPISA for various test problems. While comparing ES ( $\mu + \lambda$ ) and MDE it is found that for Problem-1 & 2 *NFE* for MDE is about 53.56% and 78.27% less than that of ES ( $\mu + \lambda$ ) but for Problem-3, 5, and 6, *NFE* for ES ( $\mu + \lambda$ ) is, respectively, about 11.48%, 43.68%, and 53.85% less than that of MDE. It is important to note that ES ( $\mu + \lambda$ ) is found to converge to a non-optimal solution for Problem-4\* & 7, whereas MDE is able to locate the global optimum for all the problems solved in the present study.

**Table-4.8. Comparison of MDE, GA, ES, M-SIMPISA & M-SIMPISA-pen**

Problem No.	<i>NFE/NRC</i>				
	GA	M-SIMPISA	M-SIMPISA-pen	ES	MDE
1	6787/100	607/99	16282/100	1518/100	705/100
2*	13939/100	10582/83	14440/100	2255/100	490/100
3	107046/90	#/0	38042/100	1749/100	1974/100
4*	22489/100	14738/100	42295/100	**/0	1797/100
5	102778/60	22309/60	63751/97	6710/100	11914/100
6	37167/100	27410/87	33956/95	2536/100	5495/100
7	225176/0	#/0	257536/92	**/0	40550/100

# Execution halted

\*\* Converged to a non-optimal solution

It may be noted that GA could not converge to global optimum, while execution was reported to be halted in M-SIMPISA for Problem-7. However, for Problem-7, *NRC* is 100% only for MDE, and 92% for MIMPISA-pen, while GA, M-SIMPISA, and ES ( $\mu + \lambda$ ) were not able to converge to global optimum. The performance of ES ( $\mu + \lambda$ ) and MDE is found to be comparable in terms of function evaluations but MDE has the advantage over ES in terms of convergence to optimal solution. The performance of MDE is found to be better than that of GA & M-SIMPISA in optimizing the mixed integer nonlinear programming problems considered in the present study. Also, results clearly show the potential of ES ( $\mu + \lambda$ ) and MDE for solving such process synthesis problems. We have already seen that MDE is better

than DE in terms of functions evaluations (*NFE*) and convergence to optimal solution (*NRC*).

#### 4.7. Conclusions

Seven test problems on process synthesis and design from chemical engineering have been solved using DE and MDE in the present work. Performance of various algorithms (e.g. GA, M-SIMPSA, M-SIMPSA-pen, ES ( $\mu+\lambda$ ), DE, and MDE) is compared in terms of *NFE* and *NRC*. Also, the two approaches for handling binary variables along with two methods for handling bound violations are studied and compared. Results indicate that the method without forcing the bound is robust however the CPU-time is more as compared to the forced bound method. It is important to note that for problems where global optimum is located on the bound of decision variables forced bound method is found to outperform the method without forcing the bound (e.g. Problem-7) whereas for problems where local optimum is located on the bound of decision variables the method without forcing the bound is found to be robust (e.g. Problem-3). Also, the Approach-1 for handling binary variables is found to be better than that of Approach-2. This is because of addition of constraints (as binary or discrete variable is modeled as continuous variable) and addition of nonconvexities to the problem in case of Approach-2. It is found that *NFE* is the least and *NRC* is highest in MDE as compared to GA & M-SIMPSA-pen. The *NRC* in M-SIMPSA is better than that of GA, M-SIMPSA, and ES ( $\mu+\lambda$ ). The ES ( $\mu+\lambda$ ) and MDE are comparable in terms of *NFE* (rather ES ( $\mu+\lambda$ ) perform slightly better than MDE) but MDE is found to be robust than ES ( $\mu+\lambda$ ) being higher *NRC* for Problem-4\*and 7 (Table-4.8). The performance of MDE is found to be the best among the methods compared for all the problems studied.

**HDE AND ITS APPLICATION TO TEST  
FUNCTIONS & SELECTED NON-LINEAR  
CHEMICAL PROCESSES**

**5.1. Introduction**

For an efficient global optimization method, two aspects that are important are (i) a good search strategy and (ii) fast convergence. Population based algorithms are good in first aspect, while gradient based methods show better performance in second aspect, i.e., finding optimum in convex region. Search space for general optimization problem can be divided into several convex regions and each region has a local optima. Classical gradient-based optimization methods show better performance than evolutionary algorithms (EAs) in finding the minimum of a convex region. However, they are not good at finding the global optima of the problem with multiple local optima. This drawback could be overcome by using EA in combination with classical gradient method.

In the present chapter an accelerated hybrid evolutionary algorithm is proposed exploiting the best characteristics of both evolutionary and gradient search methods

for finding global optimum. This new method is a hybrid of DE and quasi-Newton (QN) method (Hybrid Differential Evolution). The performance of the algorithm, proposed in the present study, i.e., Hybrid Differential Evolution (HDE) is compared with the performance of DE by applying these two algorithms to some benchmark test functions reported in literature (e.g. **HIM**: Himmelblau function; **GP<sub>2</sub>**: Goldstein and Price function; **ES<sub>2</sub>**: Easom function; **H<sub>3</sub>**: Hartmann function; **R<sub>2</sub>** and **R<sub>5</sub>**: Rosenbrock function; **Z<sub>2</sub>** and **Z<sub>5</sub>**: Zakharov function, subscript 2, 3, and 5 indicate dimensions of the problems chosen), and further the performance is evaluated on selected nonlinear chemical engineering processes also. Most of these test functions and chemical engineering problems already have been used for evaluation of performance of MDE in Chapter-3.

## 5.2. HDE Algorithm

The pseudo code of HDE algorithm proposed and used in the present study is as follows:

- *Input* the value of  $D$ ,  $NP$ ,  $CR$ ,  $F$ , strategy number ("strategy"  $\in \{1, 2, 3, \dots, 10\}$ ) &  $gen\_max$  and lower & upper bounds of variables ( $X_j^{(lo)}$ ,  $X_j^{(hi)}$ ).
- *Initialize* all the vector population randomly in the given upper & lower bound.  
 $count = 0$ ; ( $count$  is generation counter)  
*for*  $i \leq NP$  and *for*  $j \leq D$ :  $X_{i,j, count=0} = X_j^{lo} + rand_j[0,1] * (X_j^{(hi)} - X_j^{(lo)})$   
 End *for*.  
 Calculate *cost*.  
 End *for*.
- Find out the vector with the *lowest cost*.  
 Assign  $best_i = best_j = X_{j, count}$  and  $f_{min} = f(X_{i, count})$ .
- **Repeat:**
  1. Perform *mutation & crossover*.



For each target vector ( $X_{j,count}$ ), select three distinct vectors (select five, if two vector differences are to be used) randomly from the current population (primary array) other than target vector. Randomly select  $r1, r2, r3, \in \{1,2,3,\dots, NP\}$ ,  
except:  $r1 \neq r2 \neq r3 \neq i$ ;  $j \in \{1,2,\dots, D\}$ , randomly selected for each  $i$ .

if strategy = 1 i.e. DE/rand/1/bin

for  $k \leq D$ :

$$U_{j,count+1} = X_{r3,j,count} + F * (X_{r1,j,count} - X_{r2,j,count}) \quad \text{if } (rand_j [0, 1] < CR$$

$$\text{or } k = D) U_{j,count+1} = X_{i,j,count} \quad \text{Otherwise}$$

If bounds are violated:

$$U_{j,count+1} = X_j^{lo} + rand_j [0,1] * (X_j^{(hi)} - X_j^{(lo)})$$

if ( $U_{j,count+1} < X_j^{lo}$  or  $X_{j,count+1} > X_j^{(hi)}$ )

$$U_{j,count+1} = U_{j,count+1} \quad \text{Otherwise}$$

End for.

End if.

.  
.  
. (Till 10<sup>th</sup> strategy)

Select;

$$best_j = U_{j,count+1} \quad \text{if } f(U_{j,count+1}) \leq f_{min}$$

$$\text{Where Select is: } X_{i,count+1} = \begin{cases} U_{i,count+1} & \text{if } f(U_{i,count+1}) \leq f(X_{i,count}) \\ X_{i,count} & \text{otherwise} \end{cases}$$

End For.

$$best_{it} = best_j.$$

Find maximum and minimum value of 'f'.  $count = count+1$ ;

Call QN to improve upon  $best_{it}$ .

Terminate if  $|OF_{cal} - OF_{Anal}| \leq 10^{-6}$ . (As per desired accuracy)

Till termination criteria do not meet.

- Print results.

In HDE algorithm, the initial steps of initialization, mutation and crossover are the same as that of DE. The difference lies in selection step where best population vector is passed to QN subroutine as an initial guess. QN subroutine further improves the quality of solution and we know that it is able to locate the optimum of convex region efficiently. This optimum would be the global optimum if initial guess were in the region of global optimum. In this way the algorithm runs till termination criterion is met. The performance of this proposed HDE algorithm is compared with that of DE for benchmark test functions and chemical engineering processes in subsequent sections.

### **5.3. Test Functions**

Several multimodal benchmark test functions are used to test the reliability and efficiency of proposed HDE algorithm. The selected test functions are already discussed in detail in section-3.5 of Chapter-3.

#### **5.3.1. Results and Discussion**

A computer code is developed using MATLAB for HDE algorithm based on the pseudo code given in previous section-5.2. The reliability and efficiency of HDE is tested for several multimodal benchmark test functions (given in section-3.5 of Chapter-3) and compared the performance with that of DE. Each selected function is solved 10 times, each time with a different random number state. The reliability and efficiency of these methods is evaluated based on the following criteria: (a) Success rate (SR) of finding the global minimum (b) average number of objective function evaluations or CPU-time in seconds (c) Average absolute deviation of the best objective function value in each experiment from the true minimum. These criterions

are chosen for comparison purposes. Also, the criterion for convergence is chosen to be the same as used by the other authors for comparison purposes.

The AAD is defined as the average of  $|OF_{cal} - OF_{Anal}|$  where  $OF_{cal}$  is the objective function value at the best point found in each successful experiment and  $OF_{Anal}$  is the known global minimum. The minimization is considered successful if  $|OF_{cal} - OF_{Anal}| < (10^{-4} OF_{Anal} + 10^{-6})$  is fulfilled. All selected mathematical functions are solved by both HDE and DE algorithms. Table 5.1 shows the results obtained by both DE and HDE algorithms. The Average Absolute Deviation (AAD) and CPU-time are evaluated based on the successful experiments only. It may be noted that the global optimum obtained with HDE is same as reported in literature (Teh and Rangaiah, 2003; Chelouah and Siarry, 2000) for all the test functions. It presents the comparison, in terms of CPU-time in seconds and AAD that represent the average value per experiment over all the 10 experiments. The termination criterion used is either  $|OF_{cal} - OF_{Anal}| < (10^{-4} OF_{Anal} + 10^{-6})$  or Maximum number of generation (TMAX = 500). The key parameters used are  $NP = 10D$ ,  $CR = 0.5$ ,  $F = 0.8$ . The strategy used is DE/rand/1/bin in all the experiments. All the experiments are carried out on Matlab-5.1 installed on PIII/500 MHz/128 MB RAM. The quasi-Newton method used in HDE is the MATLAB subroutine E04JAF.

**Table-5.1. Comparison of DE with HDE**

TEST FUNCTIONS	CPU-time		AAD		SR (%)	
	DE	HDE	DE	HDE	DE	HDE
HIM	1.338	0.061	4.2E-07	7.3E-16	100	100
GP <sub>2</sub>	1.358	0.077	1.4E-04	4.2E-14	100	100
ES <sub>2</sub>	2.064	0.506	3.7E-05	1.7E-15	100	100
H <sub>3</sub>	2.084	0.141	2.6E-04	2.1E-07	100	100
R <sub>2</sub>	3.164	0.062	6.4E-07	1.4E-15	100	100
Z <sub>2</sub>	1.312	0.058	5.5E-07	5.2E-17	100	100
R <sub>5</sub>	78.800 <sup>#</sup>	2.08	0.027 <sup>#</sup>	1.4E-15	0	100
Z <sub>5</sub>	41.013	0.285	6.9E-07	1.8E-15	100	100

<sup>#</sup> Values corresponding to zero success rate

The results in Table 5.1 show that the reliability of HDE and DE is comparable, the success rate being 100% in both DE and HDE for each test function excluding one, i.e.,  $R_5$ . In case of test function  $R_5$  using DE, the success rate was found to be zero, which is due to limited TMAX, 500 in present study. After 500 generations AAD is found to be 0.027, which indicates that it requires more TMAX, i.e., greater than 500 generations to reach to the true minimum. Hence, it is the limitation on TMAX, which caused zero success rates. The AAD values for HDE are much less than the AAD values for DE. This indicates that HDE is able to locate global minimum more accurately. Moreover, CPU-time is much less than that for DE in each of the test functions indicating high efficiency and speed of HDE as compared to DE. Also, it is interesting to note that the performance of HDE is significantly better for higher dimension problems ( $R_5$  and  $Z_5$ ).

Fig. 5.1 shows the variation of Absolute Deviation ( $AD = |OF_{cal} - OF_{Anal}|$ ) using DE and HDE algorithms for test function named  $GP_2$ , for ten experiments. Variation in Fig. 5.1 clearly indicates that in each experiment, the accuracy of solution obtained is high in HDE over that obtained with DE. Also, there is no single experiment where AD is found to be more using DE than HDE, which implies that HDE performed better than DE.

Table 5.2 and Table-5.3 summarize results obtained using DE, HDE along with those reported by Teh and Rangaiah (2003) using TS-QN and GA-QN, and those of Chelouah and Siarry (2000) for Enhanced Continuous Tabu Search (ECTS). In ECTS, tabu search was implemented for both diversification and intensification. The GA program in GA-QN is based on floating-point encoding, selection via stochastic universal sampling, modified arithmetic crossover and non-uniform mutation. The

quasi-Newton method in GA-QN is same as that in TS-QN (i.e., IMSL subroutine, DBCONF).

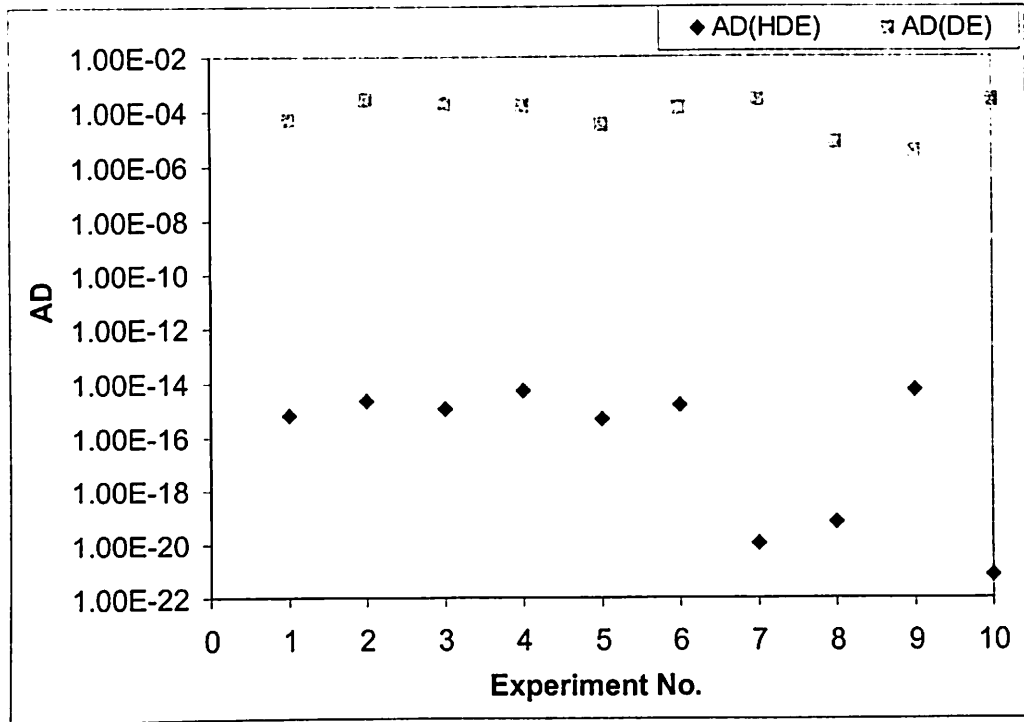


Fig. 5.1. AD variation for GP<sub>2</sub> function using for DE and HDE

Table-5.2. Comparison of AAD for various algorithms

Test Functions	Average Absolute Deviation (AAD)				
	DE	HDE	ECTS	TS-QN	GA-QN
HIM	4.2E-07	7.3E-16	---	---	---
GP <sub>2</sub>	1.4E-04	4.2E-14	0.002	7.61e-10	3.58e-12
ES <sub>2</sub>	3.7E-05	1.7E-15	0.01	1.07e-6	1.07e-6
H <sub>3</sub>	2.6E-04	2.1E-07	0.09	2.50e-6	2.57e-6
R <sub>2</sub>	6.4E-07	1.4E-15	0.02	2.19e-8	5.35e-11
Z <sub>2</sub>	5.5E-07	5.2E-17	2e-7	1.76e-8	5.00e-11
R <sub>5</sub>	0.027 <sup>#</sup>	1.4E-15	0.08	3.09e-10	3.06e-12
Z <sub>5</sub>	6.9E-07	1.8E-15	4e-6	3.91e-9	1.40e-10

# Success rate is zero

Table-5.3. Comparison of Success Rate (SR) for various algorithms

Test Functions	Success Rate (%)				
	DE	HDE	ECTS	TS-QN	GA-QN
HIM	100	100	---	---	---
GP <sub>2</sub>	100	100	100	99	100
ES <sub>2</sub>	100	100	100	85	100
H <sub>3</sub>	100	100	100	100	100
R <sub>2</sub>	100	100	100	100	100
Z <sub>2</sub>	100	100	100	100	100
R <sub>5</sub>	0	100	100	79	79
Z <sub>5</sub>	100	100	100	100	100

The results in Table-5.2 show that DE gives more accurate solution than ECTS and solution quality is comparable to that obtained from TS-QN. The HDE gives the more accurate solution than all other algorithms. ECTS gives the poor accuracy among all the algorithms shown in Table-5.2. The comparison of various algorithms according to success rate is shown in Table-5.3. The HDE and ECTS have the same or highest success rate, i.e., 100% and hence more reliable than the other algorithms. The reliability of GA-QN and DE are comparable. Strictly, TS-QN seems to be the least reliable one. In general, all the algorithms have good reliability. The results stated above clearly show the potential of HDE. In the next section, applying HDE and DE to selected nonlinear problems encountered in chemical engineering will evaluate the performance of these algorithms on real world problems.

#### **5.4. Selected Non-linear Chemical Processes**

There are many chemical processes which are nonlinear and complex with reference to optimal operating conditions with many equality and inequality constraints. Non-linearities are introduced by process equipment design relations, by equilibrium relations and by combined heat and mass balances. In this section the following processes are considered for evaluating the performance of HDE and its comparison with DE: (1) water pumping system, (2) heat exchanger network design, (3) three stage compressor with inter-cooling, and (4) drying problem. Since, QN subroutine (E04JAF) as available in MATLAB, can be used for unconstrained optimization method therefore the problems are either nonlinear unconstrained optimization problem or are reformulated as unconstrained optimization problem. The termination criterion used is  $|OF_{cal} - OF_{Anal}| < (10^{-4} OF_{Anal} + 10^{-6})$ . All the experiments are done on Matlab-5.1 installed on PIII/500 MHz/128 MB RAM.

#### 5.4.1. Water Pumping System

In the present study (Babu and Angira, 2004) modified problem as discussed in section-3.5.8 of Chapter-3, is taken as case study. First equality is used as true objective function while the other two equalities are treated as constraints. The unconstrained optimization problem, incorporating constraints into objective function, is formulated as follows:

$$\text{Min. } f = x_3 = 150 + 0.5(x_1 + x_2)^2 + R(h^2 + g^2)$$

Subject to

$$(0, 0) \leq x \leq (9.422, 5.903);$$

Where

$$h = 6x_1^2 - 30x_1 - 250.0 + 150.0 + 0.5(x_1 + x_2)^2 = 0.0 ;$$

$g = 12x_2^2 - 20x_2 - 300.0 + 150.0 + 0.5(x_1 + x_2)^2 = 0.0$ ; and  $R (=10^{10})$  is penalty on constraint violation. The global optimum obtained is:  $(x; f) = (6.293429, 3.821839; 201.159334)$ .

##### 5.4.1.1. Results and Discussion

Table 5.4 shows the results obtained by both DE and HDE. It may be noted that the global optimum is same as reported in literature (Stoecker, 1971; Leibman et al., 1986; Ryoo and Sahinidis, 1995), i.e.,  $f = 201.159334$  and the flow rates through Pump-1 & Pump-2 are  $x_1 = 6.293430$  &  $x_2 = 3.821839$  respectively.

Table-5.4 presents the comparison, in terms of CPU-time in seconds and Average Absolute Deviation (AAD). In this table, CPU-time and AAD represents the average value per experiment over all the 10 experiments and key parameters used are  $NP = 10 * D$ ,  $CR = 0.5$ ,  $F = 0.8$ . The strategy used is DE/rand/1/bin in all the experiments. It is clear from AAD values shown in Table-5.4 that HDE is able to locate the global

optimum with a high accuracy as compared to DE. Also the CPU-time taken by HDE is very much less, i.e., only about 3% of that taken by DE. The success rate is 100% in both the algorithms.

**Table-5.4. Comparison of DE with HDE for WPS Problem**

Methods	(CPU-time)	AAD	SR (%)
DE	2.992	9.2E-03	100
HDE	0.088	6.1E-08	100

Fig. 5.2 shows the variation of Absolute Deviation ( $AD = |OF_{cal} - OF_{Anal}|$ ) for DE and HDE for WPS problem. Fig. 5.2 clearly indicates that in each experiment, the accuracy of solution obtained using HDE is higher than that of obtained with DE. It fluctuates in between 0.005 to 0.018 for DE while for HDE it is  $6.0864 \times 10^{-8}$  for all experiments. Also, there is no single experiment where AD is found to be more using DE than HDE, which implies that HDE performed better than DE in locating optimum with high accuracy.

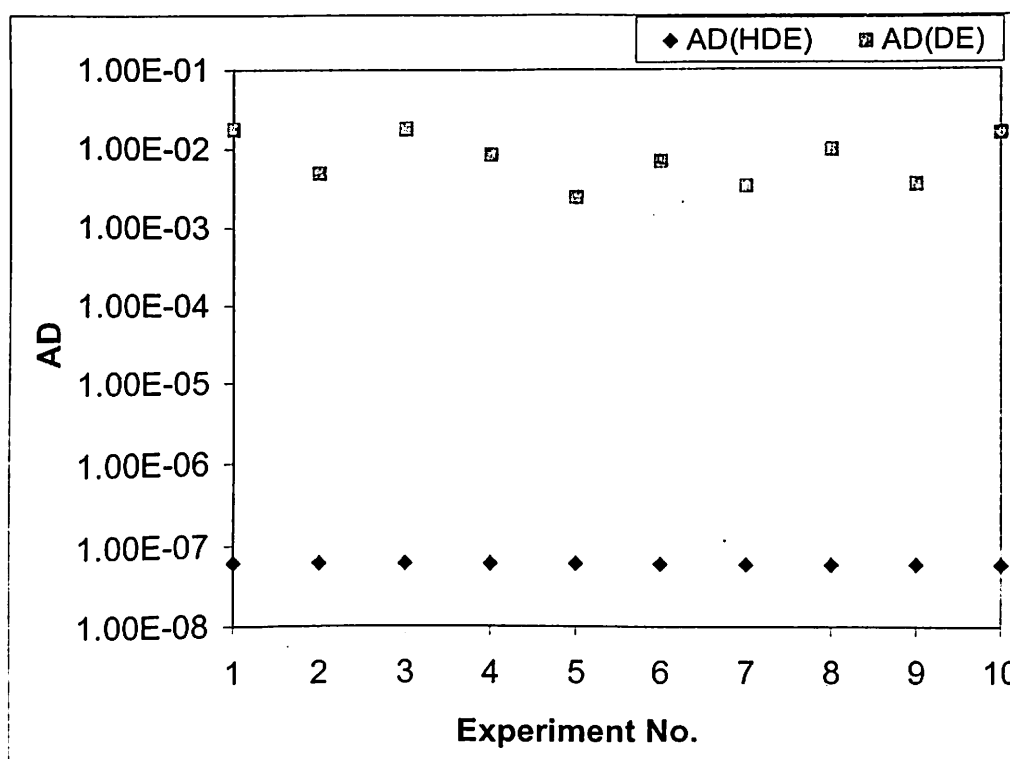


Fig. 5.2. AD variation for WPS problem using DE and HDE



### 5.4.2. Heat Exchanger Network Design (HEND)

This problem has already been discussed in Chapter-3. The problem formulation is same as that given in Ryoo and Sahinidis (1995) and is different from that described in Chapter-3. The objective is to minimize the area of the heat exchanger network as shown in Chapter-3 (Fig. 3.5.23).

$$\text{Min. } f = x_1 + x_2 + x_3$$

Subject to the following constraints:

$$100000(x_4 - 100) - 120x_1(300 - x_4) = 0.0;$$

$$100000(x_5 - x_4) - 80x_2(400 - x_5) = 0.0;$$

$$100000(500 - x_5) - 40x_3(600 - 500) = 0.0;$$

$$(0, 0, 0, 100, 100) \leq x \leq (15834, 36250, 10000, 300, 400)$$

The global optimum is:  $x = (579.306743, 1359.9712666, 5109.971263, 182.017600, 295.601150)$  with  $f = 7049.249$ .

#### 5.4.2.1. Problem Reformulation

In this problem, as can be seen above, we have five variables and three equality constraints. By eliminating three variables namely  $x_3$ ,  $x_4$ , and  $x_5$  only two independent variables ( $x_1$  and  $x_2$ ) are left to be determined. Substituting the value of  $x_4$  from first constraint into second, further substituting  $x_5$  from second constraint into third leads to evaluation of  $x_3$  in terms of  $x_1$  and  $x_2$ . In this way the number of constraints as well as number of variables are reduced.

The reformulated problem is as follows:

$$\text{Min. } f = x_1 + x_2 + x_3.$$

Subject to the following boundary constraints:

$$(100, 100) \leq x \leq (10000, 10000);$$

$$\text{Where } x_3 = 12500 - 25 \left( \frac{3200x_2 + 3.84x_1x_2 + 100 * (36x_1 + 10^4)}{(100 + 0.12x_1)(100 + 0.08x_2)} \right);$$

#### 5.4.2.2. Results and Discussion

Results obtained using both DE and HDE algorithms are shown in Table 5.5. It may be noted that the global optimum is same as reported in literature (Ryoo and Sahinidis, 1995; Adjiman et al., 1998b;) and Chapter-3 of present study, i.e.,  $f = 7049.249$ ; with areas of heat exchangers,  $x_1 = 579.306743$ ,  $x_2 = 1359.9712666$ , and  $x_3 = 5109.971263$  respectively.

**Table-5.5. Comparison of DE with HDE for HEND problem**

Methods	CPU-time	AAD	SR (%)
DE	0.253	0.3038	100
HDE	0.06	$2.7248 \times 10^{-4}$	100

Table-5.5 presents the comparison, in terms of CPU-time in seconds and Average Absolute Deviation (AAD). In this table, CPU-time and AAD represents the average value per experiment over all the 10 experiments and key parameters used are  $NP = 10 * D$ ,  $CR = 0.8$ ,  $F = 0.5$ . The strategy used is DE/rand/1/bin in all the experiments. It is clear from AAD values shown in Table-5.5 that HDE is able to locate the global optimum with a high accuracy as compared to DE. Also the CPU-time taken by DE is very large, i.e., about 4.21 times more than that taken by HDE. However the success rate is 100% in both the algorithms.

Fig. 5.3 shows the variation of AD using DE and HDE for HEND problem. Fig. 5.3 clearly indicates that in each experiment, the accuracy of solution (AD) obtained using HDE is much higher than that of obtained using DE. It ranges in between 0.0102 to 0.6404 for DE while for HDE it is  $2.7248 \times 10^{-4}$  for all the ten experiments. Also, there is no single experiment where AD is found to be more using DE than

HDE, which implies that HDE performed better than DE in locating optimum with high accuracy.

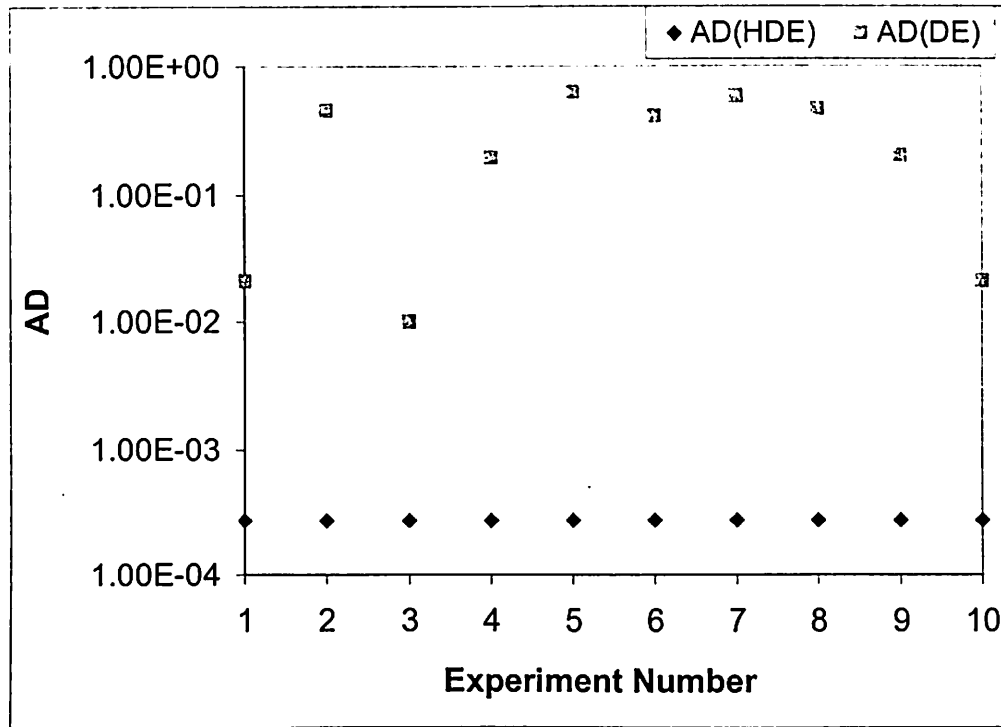


Fig. 5.3. AD variation for HEND problem using DE and HDE

### 5.4.3. Three Stage Compressor with Inter-cooling

This problem involves the determination of Minimum Work of Compression (MWC) for ideal compressible adiabatic flow (Fig. 5.4) using two different optimization techniques, i.e., DE and HDE. Most real flows lie somewhere between adiabatic and isothermal flow. For adiabatic flow, the case considered here, one couldn't establish a priori the relationship between pressure and density of the gas because the temperature is unknown as a function of pressure or density; hence the relation between pressure and density is derived using the mechanical energy balance.

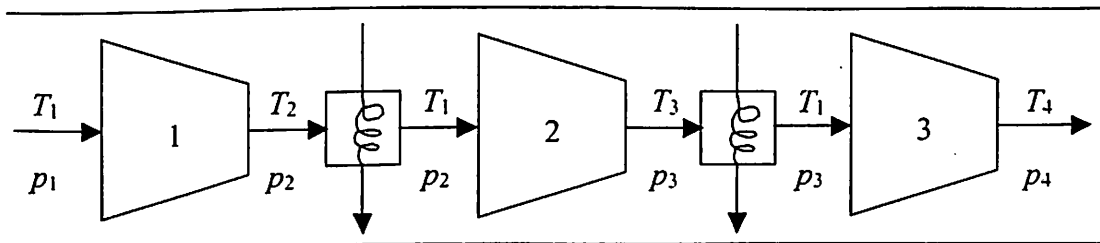


Fig. 5.4. Three stage compressor with inter-cooling.

If the gas is assumed to be ideal, and  $k = C_p/C_v$  is assumed to be constant in the range of interest from  $p_1$  to  $p_2$ , one can make use of the well-known relation given below:

$$pV^k = \text{Constant}$$

for calculating the theoretical work per mole (or mass) of gas compressed for a single-stage compressor (McCabe et al., 1993)

$$\hat{W} = \frac{kRT_1}{k-1} \left[ \left( \frac{p_2}{p_1} \right)^{\frac{k-1}{k}} - 1.0 \right]$$

where  $T_1$  is the inlet temperature and  $R$  the ideal gas constant. For a three-stage compressor with inter-cooling back to  $T_1$  between stages as shown in Fig. 5.4, the work of compression from  $p_1$  to  $p_4$  is:

$$\hat{W} = \frac{kRT_1}{k-1} \left[ \left( \frac{p_2}{p_1} \right)^{\frac{k-1}{k}} + \left( \frac{p_3}{p_2} \right)^{\frac{k-1}{k}} + \left( \frac{p_4}{p_3} \right)^{\frac{k-1}{k}} - 3.0 \right]$$

#### 5.4.3.1. Problem Formulation

The objective is to determine the optimal inter-stage pressures  $p_2$  and  $p_3$  to minimize the work of compression ( $\hat{W}$ ) keeping  $p_1$  and  $p_4$  fixed. Therefore, the objective function is given by:

$$\text{Min. } f = \hat{W} = \frac{kRT_1}{k-1} \left[ \left( \frac{p_2}{p_1} \right)^{\frac{k-1}{k}} + \left( \frac{p_3}{p_2} \right)^{\frac{k-1}{k}} + \left( \frac{p_4}{p_3} \right)^{\frac{k-1}{k}} - 3.0 \right]$$

Subject to the following boundary constraints:

$$(100, 100) \leq p_1, p_2 \leq (1000, 1000)$$

### 5.4.3.2. Results and Discussion

Let the gas be air so that  $k = 1.4$ ,  $p_1 = 100$  kPa, and  $p_4 = 1000$  kPa. Edgar and Himmelblau (2001) solved this problem using BFGS algorithm and obtained a value of inter-stage pressures  $p_2 = 215.44$  and  $p_3 = 464.17$ . Results of MWC problem using DE and HDE are shown in Table-5.6 and Fig. 5.5. Table-5.5 shows the results obtained using DE and HDE algorithms. Comparison is made in terms of CPU-time, AAD, and success rate. It is clear from Table-5.6 that DE takes almost three fold more CPU-time as compared to HDE. Also, the AAD value indicates that HDE is able to locate the optimum solution with high accuracy as compared to DE. The success rate is found to be 100% in both the algorithms.

**Table-5.6. Comparison of DE with HDE for MWC problem**

Methods	CPU-time	AAD	SR (%)
DE	0.319	$3.5587 \times 10^{-5}$	100
HDE	0.109	$1.0309 \times 10^{-6}$	100

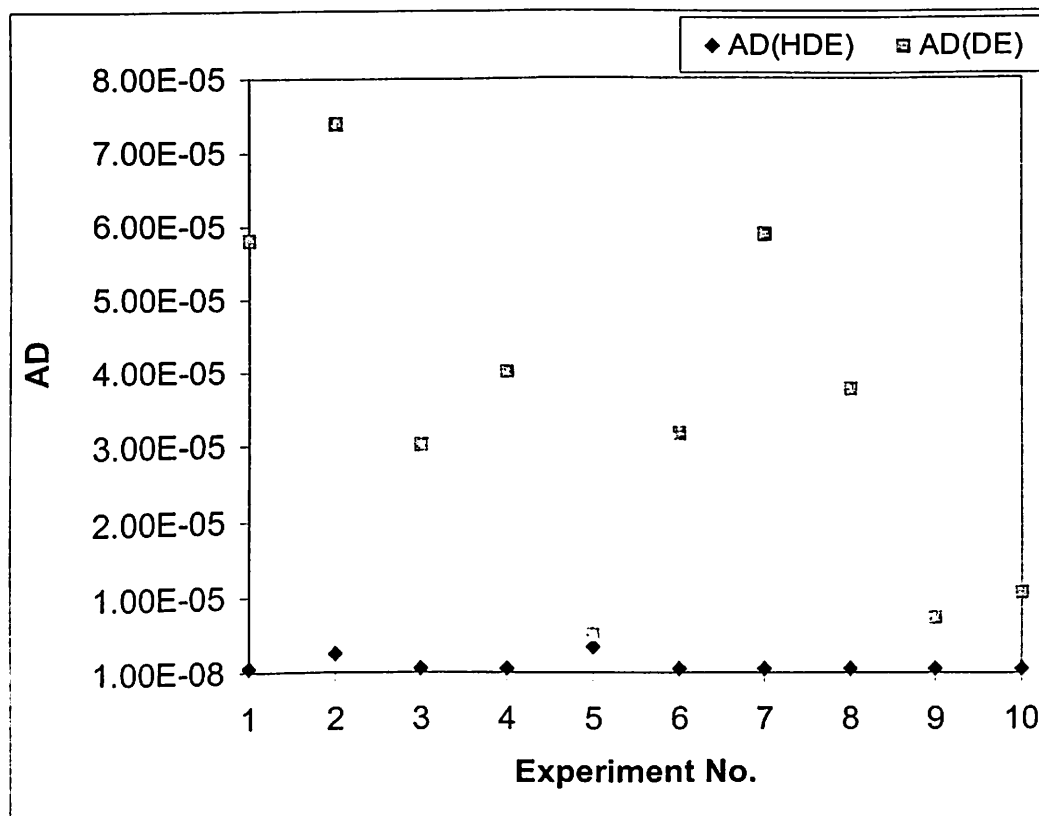


Fig. 5.5. AD variation for MWC problem using DE and HDE

Fig. 5.5 shows the variation of AD using DE and HDE for MWC problem. Fig. 5.5 clearly indicates that in each experiment, the accuracy of solution (AD) obtained using HDE is higher than that of obtained using DE. Using DE, the variation in AD values is quite high as compared to variation in AD values using HDE. For DE, it ranges from  $5.16 \times 10^{-6}$  to  $7.41 \times 10^{-5}$  while for HDE it is from  $5.36 \times 10^{-7}$  to  $3.44 \times 10^{-6}$  in all the ten experiments. Also, there is no single experiment where AD is found to be more using DE than HDE, which implies that HDE performed better than DE in locating optimum with high accuracy.

#### 5.4.4. Drying Problem

The problem has already been discussed in detail in Chapter-3. Edgar and Himmelblau (2001) gave a different formulation from that described in Chapter-3.

The problem (as described in Edgar and Himmelblau, 2001) is briefly stated below:

$$\text{Max. } f = 0.0064x_1 \left[ 1 - \exp(-0.184x_1^{0.3}x_2) \right]$$

Subject to the following constraints:

$$(3000 + x_1)x_1^2x_2 = 1.2 \times 10^{13};$$

$$\exp(0.184x_1^{0.3}x_2) = 4.1;$$

Solution reported is:  $x = (31766, 0.342)$  with  $f = 153.71$ .

##### 5.4.4.1. Results and Discussion

Table-5.7 shows the results obtained using DE and HDE algorithms. Comparison is made in terms of CPU-time, AAD, and success rate. It is clear from Table-5.7 that CPU-time taken by HDE is 56% less than that of DE. Also, the AAD value indicates that solution with high accuracy is obtained using HDE algorithm as compared to DE. It is also important to note that success rate is 90% for DE while HDE algorithm is able to locate the optimum solution with 100% success rate.

**Table-5.7. Comparison of DE with HDE for drying problem**

Methods	CPU-time	AAD	SR (%)
DE	2.833	0.02403	90
HDE	1.234	0.00351	100

Fig. 5.6 shows the variation of AD using DE and HDE for drying problem. Fig. 5.6 clearly indicates that in each experiment, the accuracy of solution (AD) obtained using HDE is higher than that of obtained using DE excepting one. For DE, it ranges from the minimum of 0.0018 to the maximum of 0.1567 while for HDE it is from the minimum of 0.0014 to the maximum of 0.0048 in all the ten experiments.

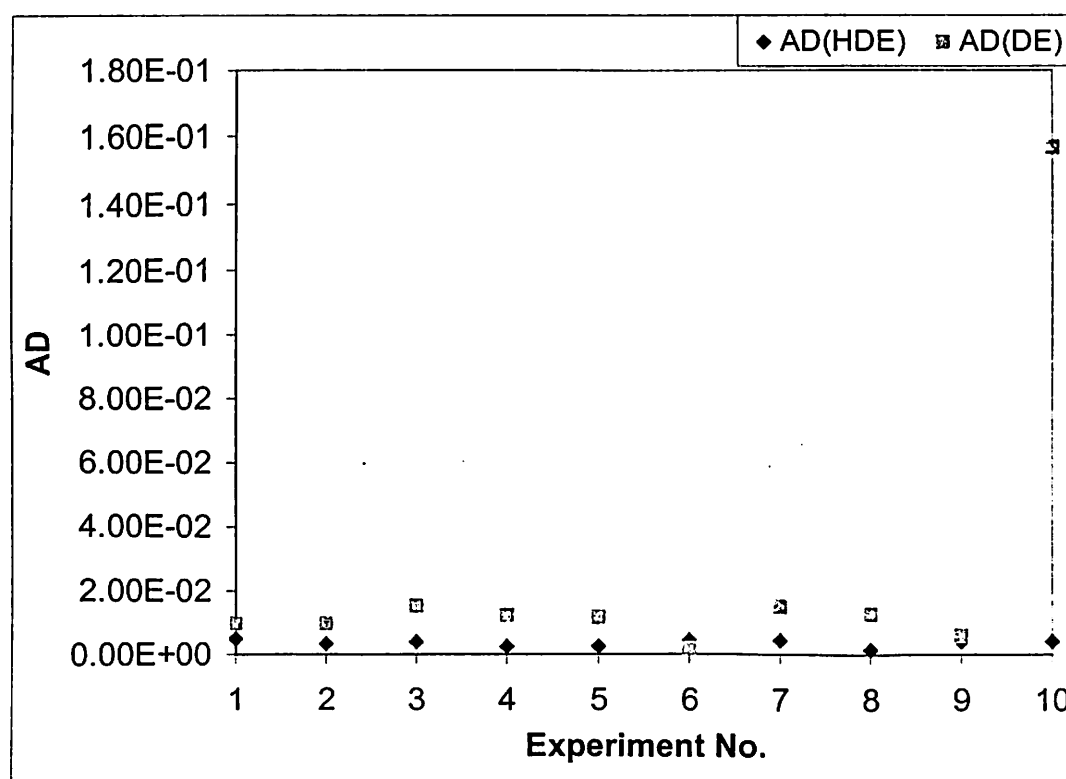


Fig. 5.6. AD variation for drying problem using DE and HDE

## 5.5. Conclusions

In this chapter a hybrid differential evolution (HDE) algorithm is proposed and tested on several benchmark test functions and further evaluated for selected nonlinear chemical engineering problems. A comparison of HDE with DE is made in

terms of CPU-time, AAD, and success rate. It is found that the two algorithms (viz., DE and HDE) are reliable in locating the global optima of the problems studied. The HDE took less CPU-time as compared to DE. Also, HDE located the global optimum with high accuracy (very low AAD values) as compared to DE. Overall, the performance of HDE is found to be better than that of DE. For high dimension problems, the performance of HDE is significantly better than that of DE. HDE is also compared with other algorithms reported in literature such as ECTS, TS-QN, and GA.



## NEW STRATEGIES OF DE AND THEIR APPLICATION TO SELECTED NON-LINEAR CHEMICAL PROCESSES

### 6.1. Introduction

As already mentioned in Chapter-3, DE is an efficient, effective and robust evolutionary optimization method. What really works behind the speed and robustness of DE is its mutation scheme which is extremely simple yet very powerful (Cormier et al., 1999). An efficient mutation scheme for real parameter optimization should have the following features: (1) use a zero mean distribution for generating mutation vectors (2) dynamically scale the distribution to suite each parameter, and (3) correlate mutations to ensure rotational invariance. Given all these demands and the indescribable variety of fitness landscapes, how can a mutation scheme be devised that will satisfy these criteria? For over thirty years, the answer provided by Evolution Strategies is 'self-adaptation'. The mutation scheme of DE is self-adaptive. Not only is the DE mutation scheme capable of dynamically scaling individual step sizes, it is also impervious to the effects of coordinate system rotation (Rudolph, 1992). Hence,

one can say that DE meets the necessary condition for optimal mutation. Still, there is a need to reduce the computational time for optimizing the computationally expensive objective functions. And therefore, an attempt to further speed up DE is considered necessary. This chapter introduces a modification to the mutation scheme of original DE that enhances the convergence rate without compromising on solution quality. Two new strategies are proposed and their performance is evaluated by comparing with that of DE using two test functions and selected nonlinear chemical engineering problems.

## 6.2. New Strategies of DE

Two new strategies (NS-1 and NS-2) are proposed and applied to two test functions and selected nonlinear chemical processes. The new strategies proposed in the present study differ from *DE/rand/1/bin* and *DE/rand/2/bin* strategies, the way the mutation and recombination is performed. The concept is quite simple and easy to understand. As we know the convergence speed of *DE/best/1/exp* strategy is highest among all the ten strategies (as seen in section-3.5.1.7 of Chapter-3) while *DE/rand/1/bin* and *DE/rand/2/bin* are robust due to their highly stochastic nature (because the search is not biased as in *DE/best/1/bin* or *DE/best/1/exp* strategies). Therefore, in order to take the advantage of both the speed and robustness, two new strategies are proposed.

In original DE's mutation scheme, the parameters in trial vector come from either random noisy vector (produced after perturbation) or from target vector depending upon the crossover constant. In the NS-1, instead of target vector, the parameter in trial vector comes from noisy vector, which is produced from perturbation of best vector found in each generation. In other words, we can say that we are taking

advantage of two existing strategies (*DE/rand/1/bin* and *DE/best/1/bin*) in one in order to increase the convergence rate while maintaining the robustness of the algorithm as shown in pseudo code.

In NS-2, the parameter in trial vector comes either from noisy vector, which is produced from perturbation of randomly chosen vector or from perturbation of best vector found in each generation. The perturbation of randomly chosen vector is done using two vector differentials with scaling factor of  $F$  and  $(1-F)$ . This is different from *DE/rand/2/bin* strategy where two vector differentials are scaled with  $F$  only rather than  $F$  and  $(1-F)$  as in present study. In NS-2, we are taking the advantage of both the *DE/rand/2/bin* and *DE/best/1/bin* strategies of DE in order to reduce the computational efforts without affecting the robustness. The pseudo codes for mutation and recombination in *DE/rand/1/bin*, NS-1 and NS-2 are given below:

(i). For *DE/rand/1/bin*

```

{
  If ( $\text{rand}_j [0,1] < CR \vee k = D$ )
  {
     $\text{trial}[j] = x[c][j] + F*(x[a][j] - x[b][j]);$ 
  }
  else  $\text{trial}[j] = x[i][j];$ 
   $j = (j+1) \% D;$ 
  /* % = modulo; index j runs from 0 to D-1 */
}

```

(ii). For NS-1

```

{
  If ( $\text{rand}_j [0,1] < CR \vee k = D$ )
  {
     $\text{trial}[j] = x[c][j] + F*(x[a][j] - x[b][j]);$ 
  }
  else  $\text{trial}[j] = \text{bestit}[j] + F*(x[a][j] - x[b][j]);$ 
   $j = (j+1) \% D;$ 
}

```

(iii). For NS-2

```

{
  If ( $\text{rand}_j [0,1] < CR \vee k = D$ )
  {
     $\text{trial}[j] = x[c][j] + F*(x[a][j] - x[b][j]) + (1-F)*(x[d][j] - x[e][j]);$ 
  }
}

```

```

}
else trial[j]=bestit[j]+F*(x[a][j]-x[b][j]);
j=(j+1)%D;
}

```

The term  $rand_j[0,1]$  represents a uniformly distributed random variable that ranges from zero to one. The subscript  $j$  indicates that a new random value is generated for each value of  $j$ ; that is, for each object variable. The vector  $bestit[j]$  is the best vector of the previous generation and the indices  $a, b, c, d, e \in \{1, 2, \dots, NP\}$  are randomly chosen population indices that are mutually different and also different from  $i$ . DE algorithm employs both mutation and recombination to create one child or trial vector  $trial[j]$ , for each parent vector  $x[i][j]$  as shown in pseudo code for *DE/rand/1/bin*, NS-1 and NS-2.

### 6.3. Test Functions

Two test functions that are known to be densely multimodal are chosen to demonstrate and test the reliability & efficiency of proposed new strategies of DE viz., NS-1 and NS-2. The first test function is so called Ackley's function, and can be found in Bäck (1996) and Fan and Lampinen (2003). This is a continuous, highly nonlinear function that causes the search with moderate complications.

*Ackley's function:*

$$f_1 = -20 \cdot \exp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}\right) - \exp\left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)\right) + 20 + e;$$

$$-20 \leq x_i \leq 30, n = 30.$$

The global minimum is:  $f_1 = 0$  with  $x_i = 0, i = 1, 2, \dots, n$ .

The second test function is commonly called Rastrigin's function, and can be found in Muhlenbein et al. (1991). This function is also considered to be relatively difficult to

minimize because the number of local optimum solutions is high. These two test functions are defined as follows:

*Rastrigin's function:*

$$f_2 = 2n + \sum_{i=1}^n (x_i^2 - 2 \cos(2\pi x_i));$$

$$-5.12 \leq x_i \leq 5.12, n = 20.$$

The global minimum is:  $f_2 = 0$  with  $x_i = 0, i = 1, 2, \dots, n$ .

### 6.3.1. Results and Discussion

First the control parameters ( $CR$  and  $F$ ) are tuned for the new strategies using the above mentioned test functions. For tuning the control parameters ( $CR$  and  $F$ ), following setting is maintained:  $gen\_max = 250$  and  $400$  respectively for Ackley and Rastrigin's function, number of independent experiments ( $NRUN$ ) =  $20$ , and  $NP = 10 * D$ . After tuning the control parameters, the performance of both the strategies is compared with that of *DE/rand/1/bin*.

#### 6.3.1.1. New Strategy-1

Fig. 6.1 shows the variation of  $CR$  vs. function value for different values of  $F$  ( $0.1 \leq F \leq 1.0$ ). Each point on the figure represents the average value of twenty experiments. It is easy to determine the best control parameters from this figure. Best control parameters are those that lead to lowest function value as the maximum number of generations is fixed.

As is evident from the Fig. 6.1, for Ackley function, the best control parameters are found to be  $F = 0.4$  with  $CR = 0.5$ . Some other combinations that lead to acceptable low function value ( $6 \times 10^{-6}$ ) are, e.g.  $F = 0.3$  with  $CR = 0.7$  and,  $F = 0.5$  with  $CR = 0.3$ . As shown in Fig. 6.2, for Rastrigin's function, the best control

parameters are found to be  $CR = 0.7$  and  $F = 0.4$ . The other possible good combination of control parameters is  $F = 0.3$  with  $CR = 0.9$ .

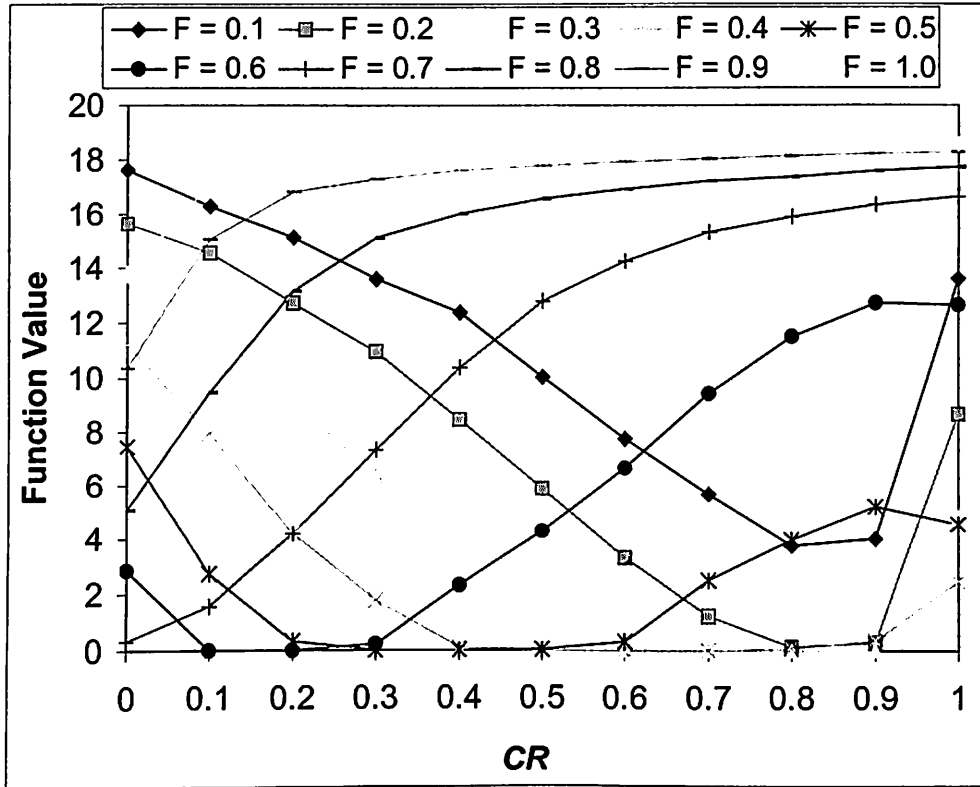


Fig. 6.1. Effect of  $CR$  and  $F$  on Ackley's function using NS-1

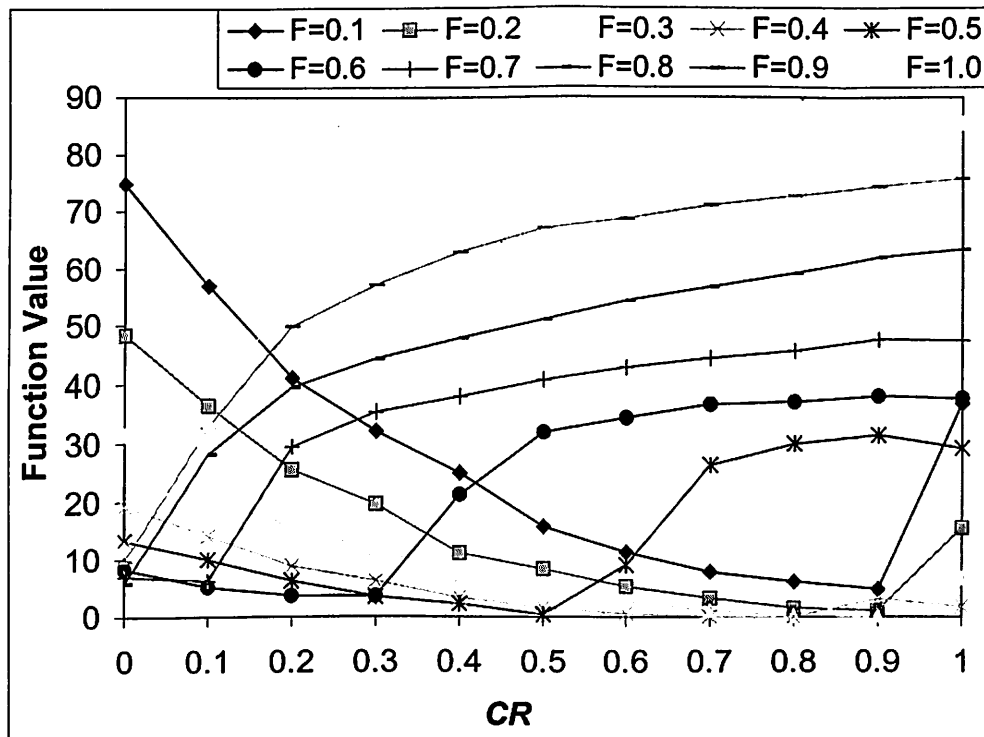


Fig. 6.2. Effect of  $CR$  and  $F$  on Rastrigin's function using NS-1

Table-6.1 shows the various possible combinations of best control parameters for the two test functions. It is clear that the choice of best control parameter is more in case of Ackley function as compared to Rastrigin's function. However, it is to be noted that the choice of best parameter is problem dependent.

**Table-6.1. Control parameters for two test functions**

Ackley			Rastrigin		
Function Value	$F$	$CR$	Function Value	$F$	$CR$
0.000003	0.3	0.7	0.048764	0.3	0.9
0.000001	0.4	0.5	0	0.4	0.7
0.000006	0.5	0.3	0.292681	0.5	0.5

From the discussion, it is found that a value of  $F$  varying from 0.3 to 0.5 is suitable to start with. However, the  $CR$  value is to be adjusted depending upon the type of problem at hand. For the Rastrigin's function  $CR$  is to be chosen 0.7 or more but for Ackley's function a value less than 0.5 may be suitable as shown in Table-6.1. Also, it is clear that for each value of  $F$  in Table-6.1,  $CR$  value is found to be higher for Rastrigin's function than that for Ackley's function to obtain best function value. It is to be noted that at  $CR = 0.0$ , the NS-1 reduces to *DE/best/1/bin* and at  $CR = 1.0$  it reduces to *DE/rand/1/bin*. Hence we can adjust the speed and robustness depending on the nature of problem at hand. Therefore the possible best control parameters for NS-1 are  $0.3 \leq F \leq 0.5$  and  $CR$  depends upon the nonlinearity of the problem.

Let us take the control parameters for NS-1 to be  $CR = 0.65$ ,  $F = 0.4$ . Fig. 6.3 and Fig. 6.4 show the convergence history for Ackley and Rastrigin's function respectively using NS-1 ( $CR = 0.65$  and  $F = 0.4$ ). Each point on the figure represents the average value over hundred experiments.

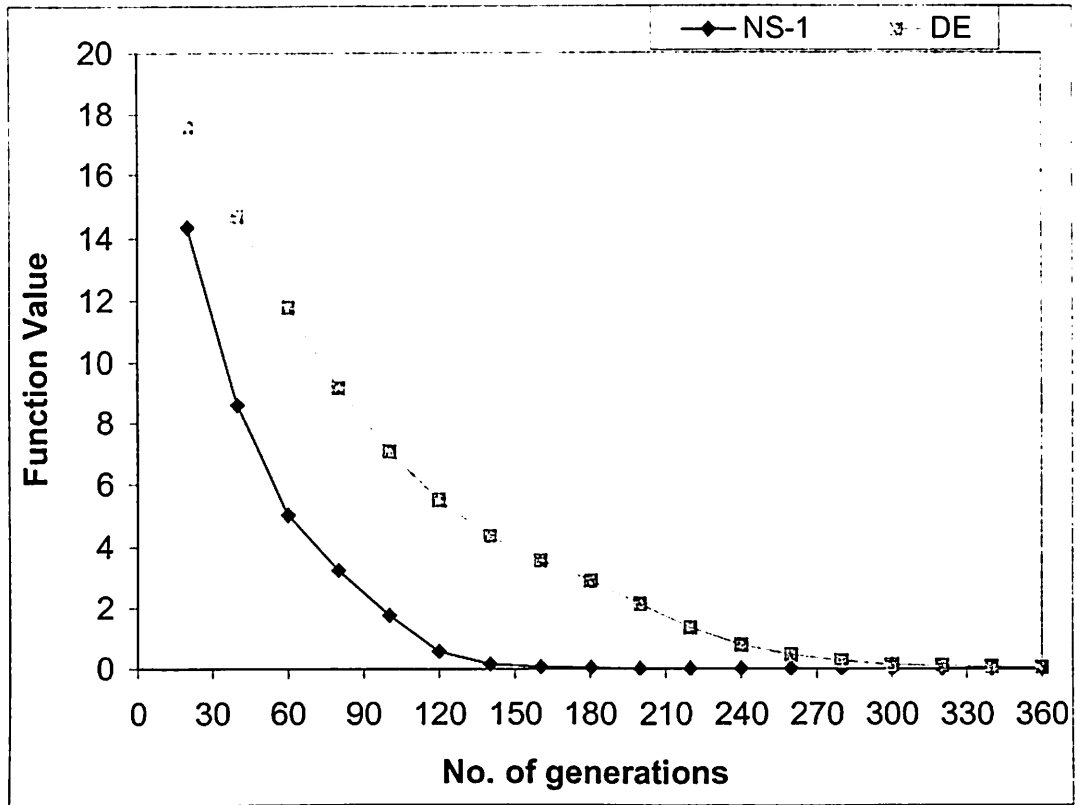


Fig. 6.3. Convergence history for Ackley's function using DE and NS-1

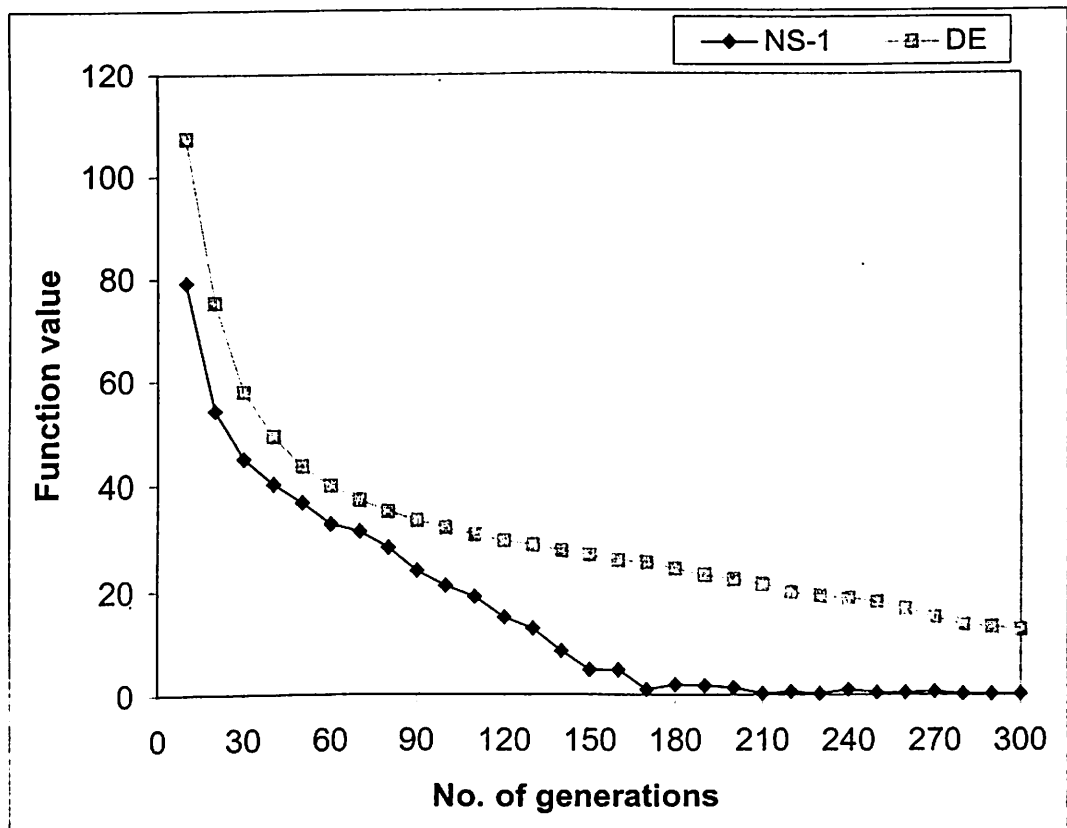


Fig. 6.4. Convergence history for Rastrigin's function using DE and NS-1



### 6.3.1.2. New Strategy-2

Figs. 6.5 and 6.6 show the effect of  $CR$  &  $F$  on Ackley's function and Rastrigin's function respectively. Each point on the figures shows the average value of twenty experiments. As shown in Fig. 6.5 for Ackley's function, at  $CR = 0.4$ , the function value is least with  $F = 0.1, F = 0.2, F = 0.3$ , and  $F = 0.4$ . Also, at  $CR = 0.3$  and  $F = 0.4$  &  $0.5$  too the function value is comparable to that obtained using  $F = 0.1, 0.2, 0.3$ , and  $0.4$ . Therefore the good combination seems to be  $CR = 0.4$  and  $F = 0.1$  or  $0.2$  or  $0.3$  or  $0.4$ .

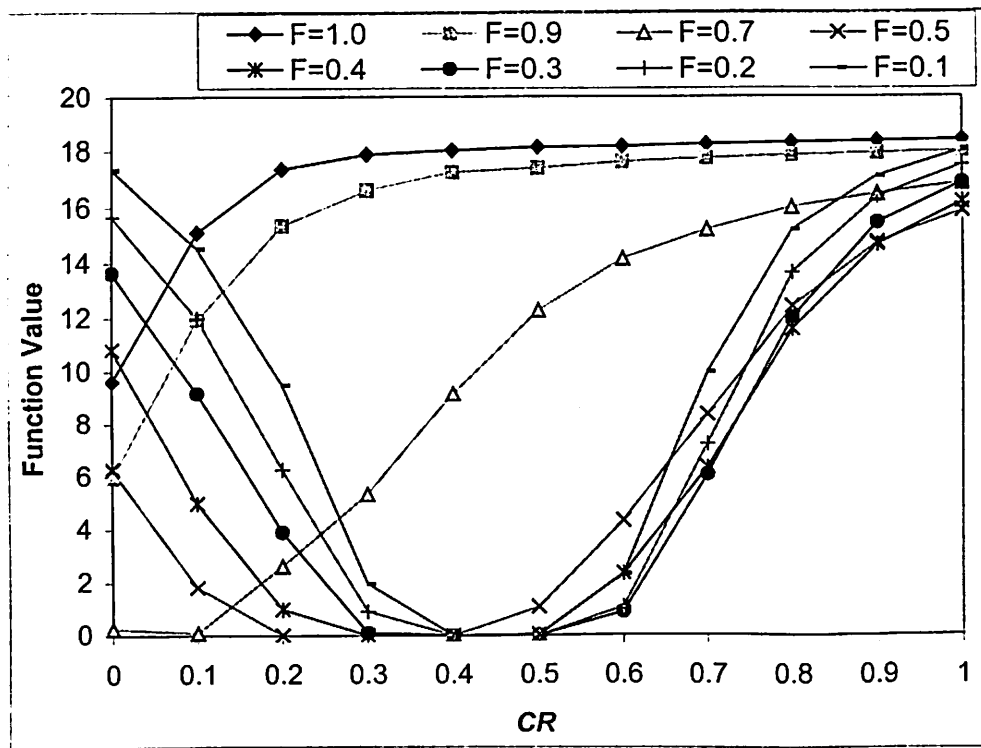


Fig. 6.5. Effect of  $CR$  and  $F$  on Ackley's function using NS-2

As shown in Fig. 6.6 for Rastrigin's function, at  $CR = 0.5$ , the function value is least with  $F = 0.1$ . However, for  $F = 0.2$ , and  $F = 0.3$  too the function value is comparable to that obtained using  $F = 0.1$ . Therefore, the best control parameters are  $CR = 0.5$  and  $F = 0.1$ . From the discussion, it is found that a value of  $F$  varying from  $0.1$  to  $0.4$  is suitable to start with. However, a  $CR$  value of  $0.4$  to  $0.5$  is found to be suitable to start with.

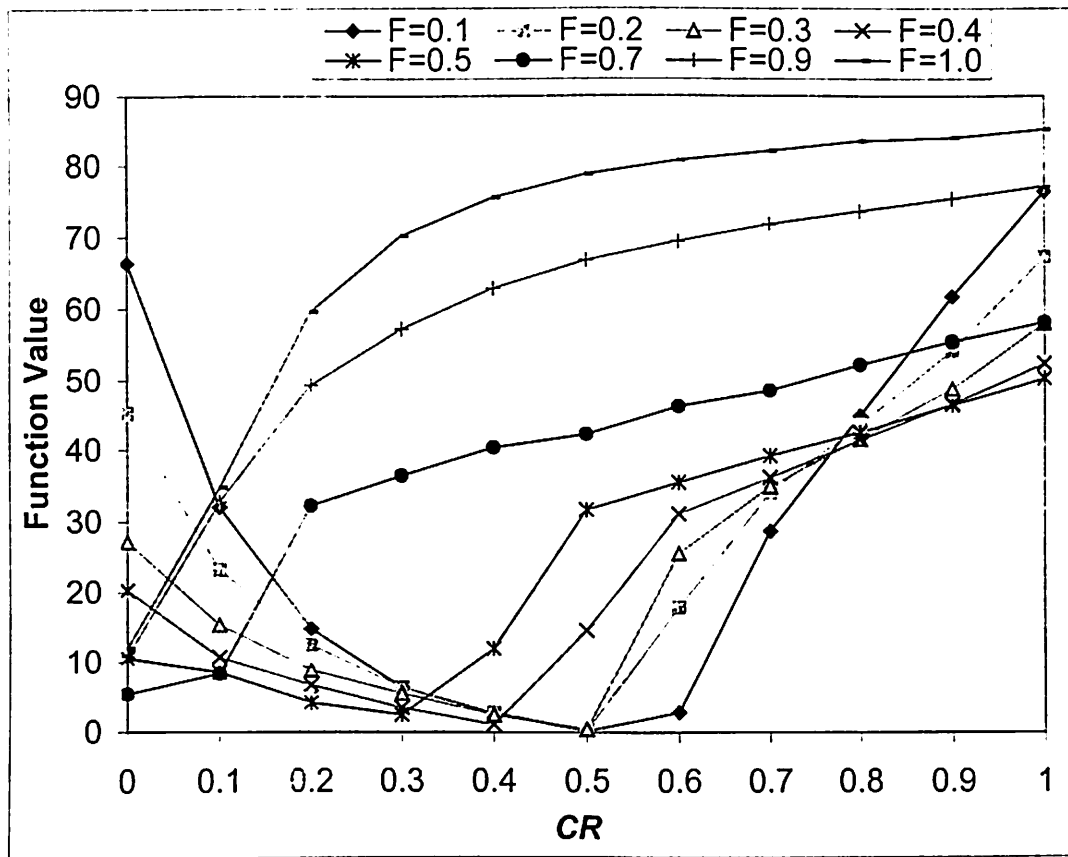


Fig. 6.6. Effect of  $CR$  and  $F$  on Rastrigin's Function using NS-2

It is to be noted that at  $CR = 0$ , it behaves like DE/best/1/bin strategy and at  $CR = 1.0$ , it reduces to DE/rand/2/bin strategy. Therefore, one can adjust  $CR$  to exploit the speed/robustness depending upon the problem encountered. Let us take the control parameters for NS-2 to be  $CR = 0.45$  and  $F = 0.1$  in the present study.

Fig. 6.7 and Fig. 6.8 show the convergence history of Ackley and Rastrigin's functions respectively. It is clear that for Ackley's function NS-2 is found to be slightly better than DE as it takes less CPU-time as compared to DE. However, the opposite is true for DE for Rastrigin's function. This clearly indicates that more experiments are further required to evaluate the performance of NS-2. In the next section, these two new strategies are applied to selected nonlinear chemical engineering problems.

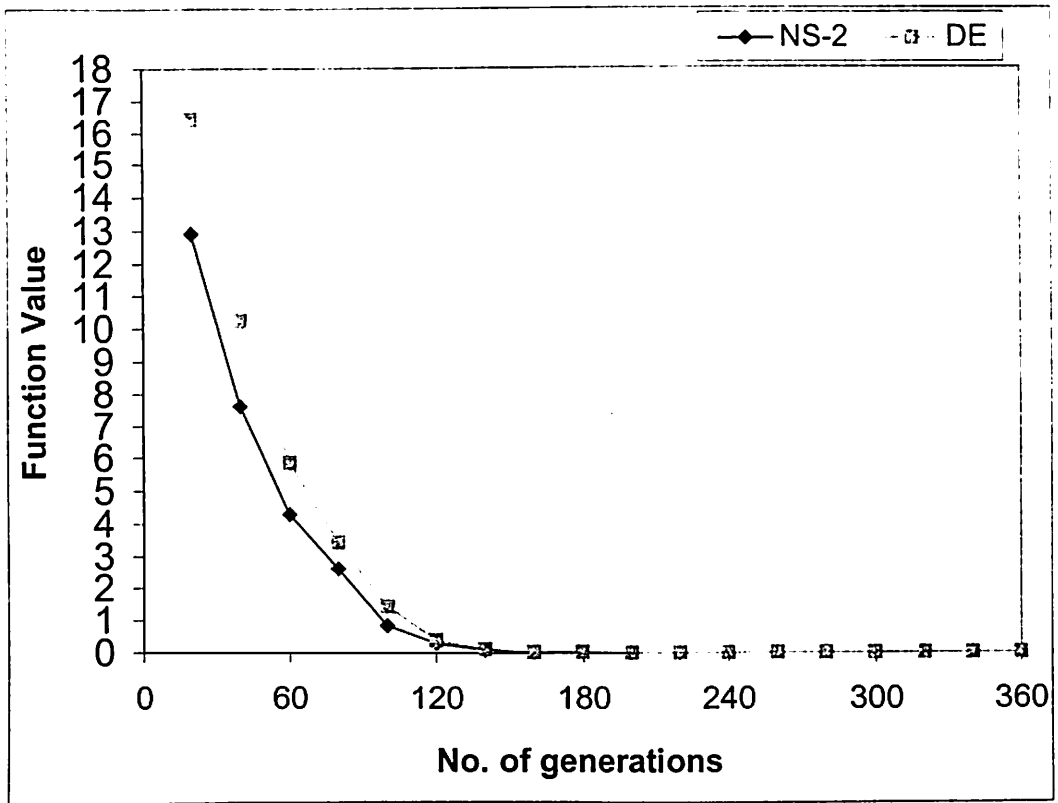


Fig. 6.7. Convergence history of Ackley function using DE and NS-2

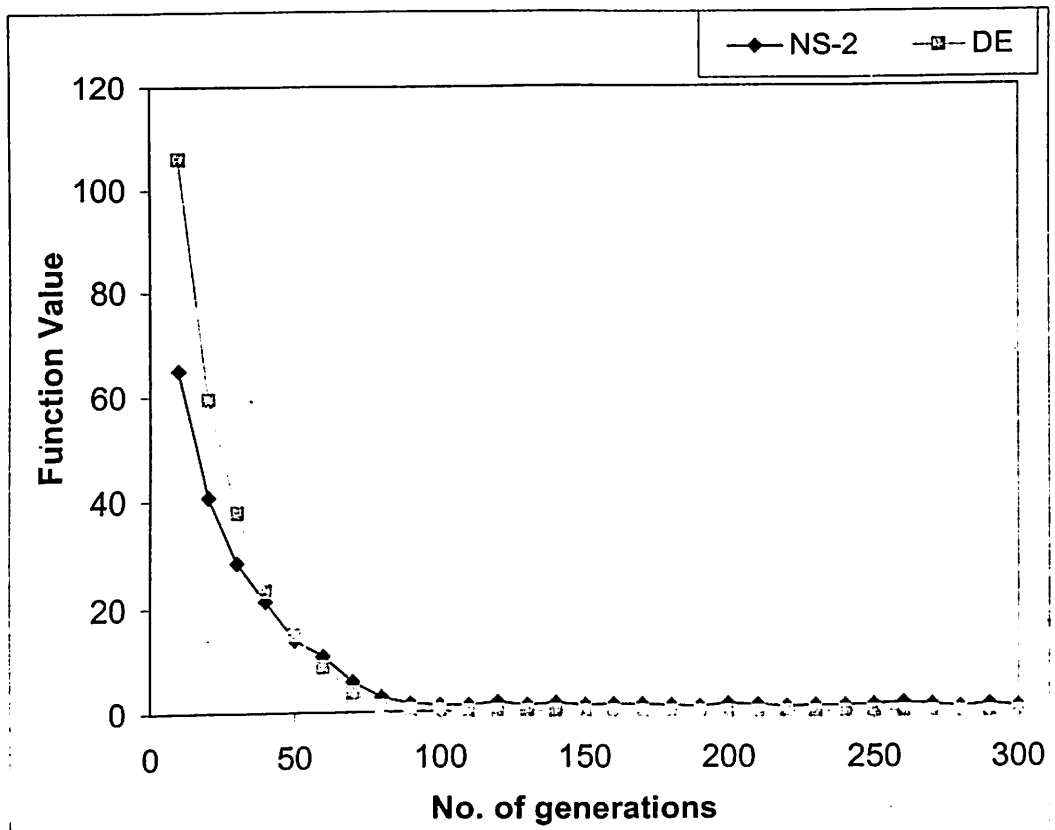


Fig. 6.8. Convergence history of Rastrigin's function using DE and NS-2

## 6.4. Selected Non-linear Chemical Processes

In the present section, three problems have been chosen to evaluate the performance of new strategies. Out of which two problems are the same as mentioned in Chapter-3. These two problems are (1) multi-product batch plant design, and (2) reactor network design. These two problems represent the difficult nonlinear problems. Third problem is dynamic optimization of a batch reactor where objective function evaluation is costly. Also, the performance of the proposed new strategies is compared with that of DE and MDE.

### 6.4.1. Multi-Product Batch Plant (MPBP)

This problem has already been discussed in section-4.5 of Chapter-4. DE and MDE were found to give local solution using method without forcing the bound. In this section new strategies are used to solve this problem and results obtained are compared with those reported in Chapter-4 using DE and MDE.

#### 6.4.1.1. Results and Discussion

Table-6.2, Table-6.3 and Table-6.4 show the results obtained using NS-1 and NS-2 with forced bound method and method without forcing the bound respectively using Approach-1. For each algorithm ten experiments are carried out with different seed values. *NFE*, *NRC* and CPU-time in the subsequent tables are the mean values of the ten experiments. The stopping criteria adopted for NS-1 and NS-2 is same as that used for MDE and DE in Chapter-4. All the experiments are carried out on Pentium-III, 500 GHz, and 128 RAM computer.

**Table-6.2. Comparison of NS-1 and NS-2 MPBP problem (FBM)**

Methods	CPU-time (s)	<i>NFE</i>	<i>NRC</i> (%)
NS-1	1.70	23640	70
NS-2	1.53	19060	70
DE	3.23	46090	100
MDE	2.846	40550	100

From Table-6.2, it is clear that *NFE* (and hence CPU-time) for NS-2 is least. However the *NRC* is 70% instead of 100 for DE and MDE. There is a saving of 10% CPU-time using NS-2 as compared to NS-1. NS-2 is found to be computationally inexpensive among all the algorithms compared in Table-6.2. *NFE* for NS-2 is about 19.37%, 58.65%, and 53.0% less than that of obtained using NS-1, DE, and MDE respectively.

Table-6.3 shows the results obtained when *CR* value is increased to 0.8 for NS-1 and to 0.6 for NS-2 in order to increase the robustness. It is found that *NRC* is 100 % for both NS-1 and NS-2. Also it is to be noted that CPU-time is still less for NS-1 and NS-2 as compared to DE and MDE although it is slightly more than that at *CR* = 0.65.

**Table-6.3. Comparison of NS-1 & NS-2 MPBP problem at higher *CR* (FBM)**

Method	CPU-time (s)	<i>NFE</i>	<i>NRC</i> (%)
NS-1	1.8623	25950	100
NS-2	2.0164	25070	100
DE	3.23	46090	100
MDE	2.846	40550	100

**Table-6.4. Comparison of NS-1 and NS-2 MPBP problem (MWFB)**

Method	CPU-time (s)	<i>NFE</i>	<i>NRC</i> (%)
NS-1	2.86	39810	0
NS-2	2.76	34010	0
DE	6.29	89490	0
MDE	5.54	79380	0

From Table-6.4, it is clear that *NFE* (and hence CPU-time) for NS-2 is least. In case of method without forcing the bound too, the NS-2 seems to be computationally efficient among all the methods compared. It is to be noted that the saving in CPU-time using NS-2 as compared to NS-1 is only 3.5% in case of method without forcing the bound instead of 10% in case of forced bound method. It is to be noted that none of the methods is able to locate the global optimum using method without forcing the bound even at high *CR* value. This is due to the fact that the global optimum is

located on the extreme (lower or upper bound) of the several decision variables. Also, the global optimum corresponds to a point where a very small variation in any of the continuous variables produces infeasibility.

#### 6.4.2. Reactor Network Design (RND)

This problem has already been discussed in section-3.4 of Chapter-3. DE and MDE are found to show poor convergencies to global optimum as it has two local optima near to global optimum solution. In this section the same problem is solved using two new strategies and results are compared with that of obtained using DE and MDE algorithms.

##### 6.4.2.1. Results and Discussion

Table-6.5 shows the results obtained using NS-1 & NS-2 and its comparison to DE & MDE algorithms. *NFE* & *NRC* represent respectively, the mean number of objective function evaluations and the percentage of experiments converged to the global optimum in all the 100 experiments (with different seed values).

**Table-6.5. Comparison of NS-1, NS-2, DE, and MDE for RND problem**

Method	CPU-time (s)	<i>NFE</i>	<i>NRC</i> (%)	<i>Key Parameters (CR/F)</i>
DE	0.041	1468	57	0.8/0.5
MDE	0.034	1253	44	0.8/0.5
NS-1	0.044	1559	65	0.65/0.4
NS-2	0.156	4048	100	0.45/0.1

All the experiments are carried out on a Pentium-III/500 GHz/128 RAM computer. RND is a difficult problem as mentioned already having two local optima near global optimum. Still NS-1 and NS-2 are able to locate the global optimum although the success rate (*NRC*) is 65 to 100% as shown in Table-6.5. In this problem, *NFE* using MDE is less by about 14.65%, 19.63%, and 69.05% than that obtained using DE, NS-1, and NS-2 respectively. It is important to note that in this problem, *NRC* using NS-2 is 100% as compared to 44%, 57%, and 65% using MDE, DE, and

NS-1 respectively as shown in Table-6.5. This clearly indicates the robustness of NS-2 strategy.

Fig. 6.9 shows the convergence history of RND problem. Each point on the graph represents an average of 100 independent experiments. It is clear from the Fig. 6.9 that error variation is least using NS-2 (0.005% to 0.0%) among all the methods compared. It is seen that error variation is more in case of MDE algorithm (0.003% to 0.014%) than DE (0.001% to 0.01%) while it is highest for NS-1 (0.005% to 0.02%). This indicates that NS-2 is best among all the methods as it is able to locate the global optimum solution with high accuracy (Fig. 6.9) and high success rate (*NRC* being 100% as shown in Table-6.5). Also, NS-2 is found to be computationally costly as compared to NS-1, DE, and MDE.

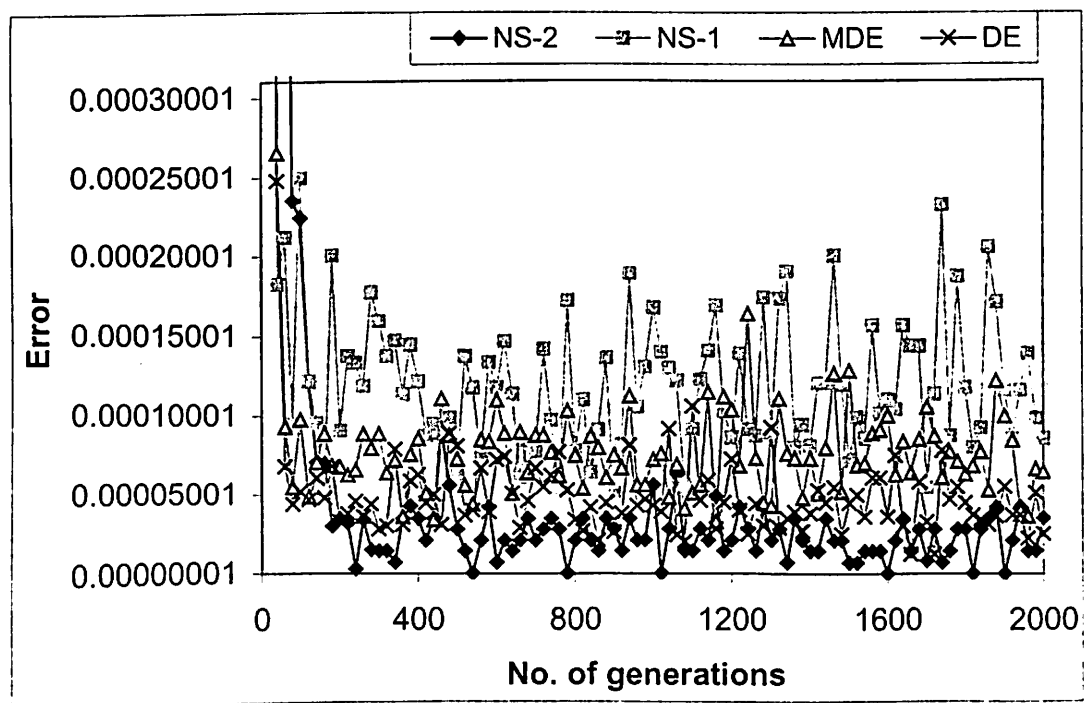


Fig. 6.9. Convergence history of RND problem

In order to enhance the robustness of the NS-1, DE, and MDE algorithms *CR* value is increased. And it is found that *CR* alone is not affecting the *NRC*. An increase in *F* value significantly affects the *NRC*. For NS-1, keeping the *CR* value at 0.65 and

increasing  $F$  to 0.7 gave a  $NRC$  value of 97%. Further, increasing the value of  $F$  to 0.9 leads to 100%  $NRC$ .

Table-6.6 shows the comparison of NS-1, DE, and MDE algorithm for the increased values of  $CR$  and  $F$ . Also, it has been found out that at  $CR = 1.0$  and  $F = 0.9$ , DE is able to achieve 100%  $NRC$  although the computation time is more than that shown in Table-6.5. MDE seems to be computationally efficient. It is to be noted that at  $CR = 0.7$  and  $F = 0.7$ , DE takes more CPU-time as compared to NS-1 but at  $CR = 1.0$  and  $F = 0.9$ , computational time is same in both the strategies. Further, the CPU-time,  $NFE$ , and  $NRC$  are not significantly affected in case of NS-1 as compared to DE.

**Table-6.6. Comparison of NS-1, DE, and MDE for RND problem**

Method	CPU-time (s)	$NFE$	$NRC$ (%)	Key Parameters ( $CR/F$ )
DE	0.069	2472	94	0.7/0.7
DE	0.060	2074	100	1.0/0.9
NS-1	0.061	2136	100	0.7/0.7
NS-1	0.059	2074	100	1.0/0.9
MDE	0.052	1860	99	1.0/0.9

### 6.4.3. Dynamic Optimization of a Batch Reactor

In this problem, we consider the consecutive reaction:  $A \xrightarrow{k_1} B \xrightarrow{k_2} C$  taking place in a batch reactor (Ray, 1981). The objective is to obtain the optimal reactor temperature progression, which maximizes the intermediate product  $B$  for a fixed batch time. The dynamics are given by the following design equations:

$$\frac{dC_A}{dt} = -k_1 C_A^2;$$

$$\frac{dC_B}{dt} = k_1 C_A^2 - k_2 C_B;$$

Where  $k_1 = 4000 * \exp\left(\frac{-2500}{T}\right)$



$$k_2 = 620000 * \exp\left(\frac{-5000}{T}\right)$$

and  $298 \leq T \leq 398$ ; with the initial conditions  $C_A(0) = 1.0$ ,  $C_B(0) = 0.0$ , and the objective function is  $J = \text{Maximize } C_B$ . We need to find out optimal temperature profile, which gives maximum intermediate product concentration. This problem has been solved by Renfro et al. (1987) using piecewise constant controls. Dadebo and Mcauley (1995) used dynamic programming for solving this problem and reported results for different number of stages.

In this problem our parameters are temperatures at different time intervals. With these temperatures we need to find out the optimal final concentration of  $B$  (by solving the above model equations along with NS-1 and NS-2).

#### 6.4.3.1. Results and Discussion

In this problem, the objective is to find out optimal temperature profile, which gives maximum intermediate product concentration. This problem has been solved by Renfro et al. (1987) using piecewise constant controls. They reported a value of 0.61 for the objective function. Logsdon and Biegler (1989) obtained a value of 0.610767. Dadebo and Mcauley (1995) used dynamic programming for solving this problem. They reported results for different number of stages. Dadebo and Mcauley (1995) reported a yield of 0.610070 for ten stages which is same as shown in Table-6.7 for NS-1 and NS-2.

**Table-6.7. Comparison of NS-1, NS-2, and DE for Batch Reactor problem**

Method	Yield	Optimal Initial Temperature $T(0)$	CPU-time (s)	NRC (%)
DE	0.610079	361.4	166.00	100
MDE	0.610079	361.4	140.00	100
NS-1	0.610079	361.4	69.18	100
NS-2	0.610079	361.4	84.31	100

In the Table-6.7, CPU-time is average of twenty experiments. It is evident that NS-1 takes about 17.95%, 50.0%, and 58.32% of CPU-time less than that of NS-2, MDE, and DE respectively. Also, it is to be noted that time saving is highly desirable in this type of problems, as the saving is of significant amount (from 15.0s to more than a minute or so) as compared to other problems where the saving is of order of 1.0s or less. NS-1 seems to be computationally efficient method among all the methods compared for the present problem. Also, it is to be noted that *NRC* is 100% in all the methods.

Fig. 6.10 shows the optimal temperature profile obtained using NS-1, NS-2 and MDE for 10 intervals of total time (i.e. for 10 stages). All the profiles are exactly same. NS-1 is able to find the optimal temperature profile faster than NS-2, DE, and MDE algorithm.

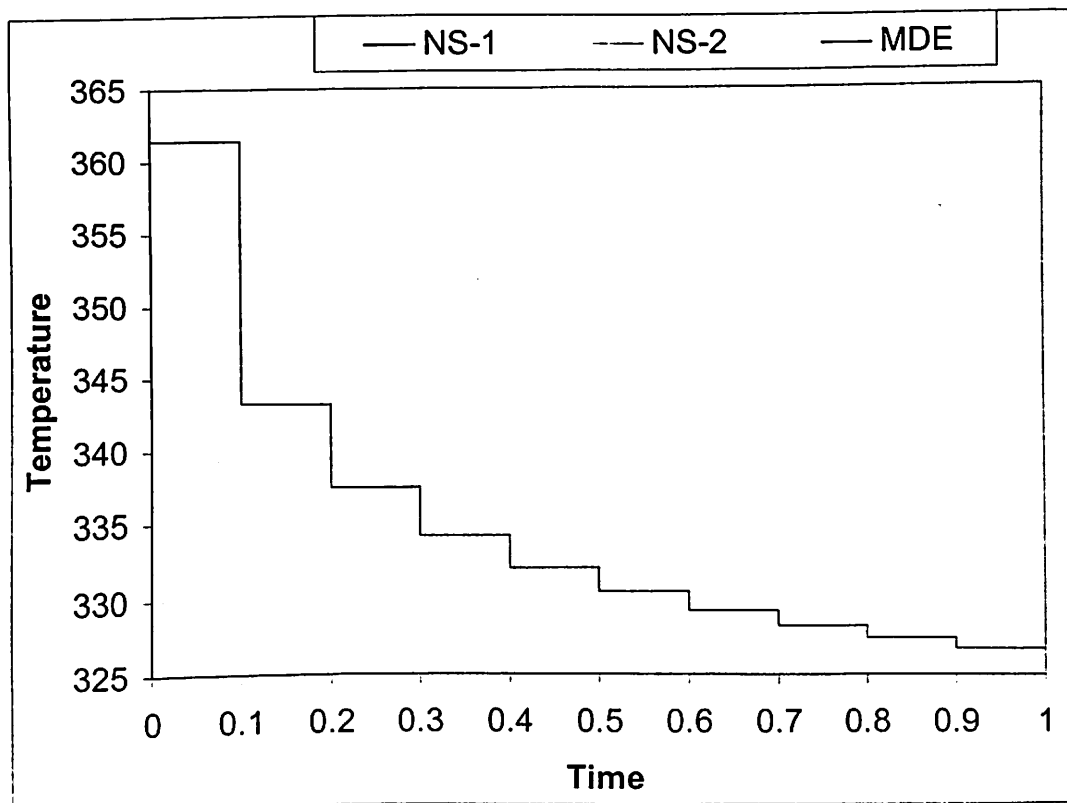


Fig. 6.10. Optimal temperature profile

## 6.5. Conclusions

In the present chapter, two new strategies are proposed and their performance is evaluated using two test functions and three nonlinear chemical engineering problems. Proposed strategies are tuned hopefully to their best control parameters. Using these tuned control parameters the three selected nonlinear chemical engineering problems have been solved. It is found that like DE and MDE, NS-1 and NS-2 also take less computation time for forced bound method as compared to method without forcing the bound. Also, none of the algorithms is able to locate the global optimum using the method without forcing the bound for the MPBP problem. NS-1 is found to be computationally efficient for the dynamic optimization of a batch reactor and RND problem as compared to NS-2. NS-2 is found to take least CPU-time for MPBP problem while MDE is found computationally efficient for RND problem. For RND problem NS-2 is found to be robust as compared to other methods. Overall, the proposed strategies (NS-1 and NS-2) are found to be competitive with DE and MDE algorithms.

## **AN EXTENSION OF DE AND MDE FOR MULTI-OBJECTIVE OPTIMIZATION**

Most of the real world optimization problems have more than one objective to be optimized, and hence they are called Multi-Objective Optimization Problems (MOOPs). Evolutionary algorithms are gaining popularity for solving MOOPs due to their inherent advantages. In this chapter, DE and MDE are extended for solving MOOPs and we call these extended algorithm as Non-dominated Sorting Differential Evolution (NSDE) and Modified Non-dominated Sorting Differential Evolution (MNSDE). The proposed algorithms are tested on two different benchmark test problems. Also, the effect of various key parameters on the performance of NSDE and MNSDE is studied.

### **7.1. Introduction**

The field of search and optimization has changed over the last few years by the introduction of a number of non-classical, unorthodox and stochastic search and

optimization algorithms. Ideally, multi-objective optimization problems require multiple trade-off solutions (a set of Pareto optimal solutions) to be found. The presence of multiple conflicting objectives makes the problem interesting to solve. Due to multiple conflicting objectives, no single solution can be termed as an optimum solution. Therefore, the resulting multi-objective optimization problem resorts to a number of trade-off optimal solutions. Classical optimization methods can at best find one solution in a single run, on the other hand evolutionary algorithms can find multiple optimal solutions in a single run due to their population based search approach.

Many engineering applications involve multiple criteria, and recently, the exploration of Evolutionary Multi-objective Optimization (EMO) techniques has increased (Coello, 1999). The ideal approach for a multi-objective problem is the one that optimizes all conflicting objectives simultaneously. Evolutionary algorithms inherently explore a set of possible solutions simultaneously. This characteristic enables the search for an entire set of Pareto optimal solutions in a single run. Additionally, evolutionary algorithms are less susceptible to problem dependent characteristics, such as the shape of the Pareto front (convex, concave, or even discontinuous), and the mathematical properties of the search space, whereas these issues are of concerns for mathematical programming techniques for mathematical tractability.

Schaffer (1985) proposed first practical approach to multi-criteria optimization using EAs, Vector Evaluated Genetic Algorithm (VEGA). After that there have been several other versions of evolutionary algorithms that attempt to generate multiple non-dominated solutions such as (Kursawe, 1991; Hajela and Lin, 1992). The concept of pareto-based fitness assignment was first proposed by Goldberg (1989), as a means

of assigning equal probability of reproduction to all non-dominated individuals in the population. Fonseca and Fleming (1993) have proposed a multi-objective genetic algorithm (MOGA). Srinivas and Deb (1995) proposed NSGA, where a sorting and fitness assignment procedure based on Goldberg's version of Pareto ranking is implemented. Horn et al. (1994) proposed Niche Pareto Genetic Algorithm (NPGA) using a tournament selection method based on Pareto dominance. Knowles and Corne (2000) proposed a simple evolution strategy (ES), (1+1)-ES, known as the Pareto Archived Evolution Strategy (PAES) that keeps a record of limited non-dominated individuals. The more recent algorithms include the (Strength Pareto Evolutionary Algorithm) SPEA (Zitzler and Thiele, 1999), NSGA-II (Deb et al., 2002), Pareto-frontier Differential Evolution (Abbass et al., 2001), and Multi-Objective Differential Evolution (Xue et al., 2003; Babu and Jehan, 2003; Babu et al., 2005a, 2005b).

Previously, a few researchers (Abbass et al., 2001; Xue et al., 2003; Babu et al., 2005a, 2005b) studied the extension of differential evolution to multi-objective optimization problem in continuous domain, but using different approach from that described in this chapter. In the present study (Angira and Babu, 2005a), NSDE, a simple extension of DE (where same mutation & crossover scheme is used as in DE, however the selection criterion is modified as it is being used for solving MOOPs) and MNSDE, an extension of MDE are proposed and tested on the two test problems. One test problem is Schaffer's function and the other is cantilever design problem.

## **7.2. Multi Objective Optimization Problems (MOOPs)**

As the name suggests, a multi objective optimization problem deals with more than one objective function. In most practical decision-making problems multiple objectives or multiple criteria are evident. Multi-Objective optimization is sometimes

referred to as vector optimization, because a vector of objectives, instead of a single objective, is optimized. General form of the multi-objective optimization problem (MOOP) is given as follows (Deb, 2001).

$$\begin{array}{lll}
 \text{Minimize / Maximize} & f_m(\mathbf{x}) & m = 1, 2, \dots, M; \\
 \text{Subject to} & g_j(\mathbf{x}) & j = 1, 2, \dots, J; \\
 & h_k(\mathbf{x}) & k = 1, 2, \dots, K; \\
 & x_i^{(L)} \leq x_i \leq x_i^{(U)} & i = 1, 2, \dots, n.
 \end{array}$$

A solution  $\mathbf{x}$  is a vector of  $n$  decision variables:  $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ . The last set of constraints are called variable bounds, restricting each decision variable  $x_i$  to take a value with a lower limit  $x_i^{(L)}$  and an upper  $x_i^{(U)}$  bound. These bounds constitute a decision variable space  $D$ , or simply the decision space. Associated with the problem is  $J$  inequality and  $K$  equality constraints. The terms  $g_j(\mathbf{x})$  and  $h_k(\mathbf{x})$  are called constraint functions. The inequality constraints are treated as ‘greater-than-equal-to’ types although a ‘less-than-equal-to’ type inequality constraint must be converted into a ‘greater-than-equal-to’ type constraint by multiplying the constraint by  $-1$ . A solution  $\mathbf{x}$  that does not satisfy all of the  $(J + K)$  constraints and all of the  $2n$  variable bounds stated above is called an infeasible solution. On the other hand, if any solution  $\mathbf{x}$  satisfies all constraints and variable bounds, it is known as a feasible solution. Therefore, we realize that in the presence of constraints, the entire decision variable space  $D$  need not be feasible. The set of all feasible solutions is called the feasible region, of  $S$ .

### 7.2.1. Pareto Optimal Solutions

A multi-objective optimization problem and its global optimal solution(s) can be defined in many ways. A solution is Pareto-optimal if it is dominated by no other feasible solution, which means that there exists no other solution that is superior at

least in case of one objective function value and equal or superior with respect to the other objective functions values (Deb, 2001).

### 7.3. Non-dominated Sorting Differential Evolution (NSDE)

NSDE algorithm is a simple extension of DE for solving multi-objective optimization problems. The working of NSDE and DE is similar except the selection operation that is modified in order to solve the multi-objective optimization problems. The detail of the NSDE algorithm is as follows:

First of all set the set the key parameters, i.e.,  $CR$  - crossover constant,  $F$  - scaling factor,  $NP$  - population size,  $Max\_gen$  – maximum number of generations of NSDE algorithm. And then randomly initialize the population points within the bounds of decision variables. After initialization of population, randomly choose three mutually different vectors for mutation and crossover operation (as is done in DE algorithm) to generate trial vector. Evaluate the trial and target vector and perform a dominance check. If trial vector dominates the target vector, the trial vector is copied into the population for next generation otherwise target vector is copied into population for next generation. This process of mutation, crossover, and dominance check is repeated for specified number of generations. Evaluate and then sort this final population to obtain the non-dominated solutions. Sorting can be done using any of the standard approaches reported in Deb (2001). In the present study, naïve and slow approach is used. In this approach, each solution  $i$  is compared with every other solution in the population to check if it is dominated by any solution in the population. If no solution is found to dominate solution  $i$ , it is member of the non-dominated set otherwise it does not belong the non-dominated set. This is how any



other solution in the population can be checked to see if it belongs to the non-dominated set.

The stopping criteria for the algorithm can be any one of the following conditions:

- (a). There is no new solution added to the non-dominated front for a specified number of generations.
- (b). Till the specified number of generations.

However, in this study, the second condition is used as termination criterion. The pseudo code of NSDE algorithm used in the present study is given below:

Set the values of NSDE parameters  $D$ ,  $NP$ ,  $CR$  and  $Max\_gen$  (maximum generations).

Initialize all the vectors of the population randomly within the bounds.

for  $i = 1$  to  $NP$

for  $j = 1$  to  $D$

$X_{i,j} = \text{Lower bound} + \text{random number} * (\text{upper bound} - \text{lower bound});$

End for

End for

Perform mutation, crossover, selection and evaluation of the objective function for trial and target vector for a specified number of generations.

While ( $gen < Max\_gen$ )

{ for  $i = 1$  to  $NP$                     */\*\* first for loop\*\*\*/*

    For each vector  $X_i$  (target vector), select three distinct vectors  $X_a$ ,  $X_b$  and  $X_c$  randomly from the current population other than the vector  $X_i$

do

    {  $r1 = \text{random number} * NP$

$r2 = \text{random number} * NP$

$r3 = \text{random number} * NP$

    } While ( $r1=i$ ) OR ( $r2=i$ ) OR ( $r3=i$ ) OR ( $r1=r2$ ) OR ( $r2=r3$ ) OR ( $r1=r3$ )

Perform mutation and crossover for each target vector  $X_i$  and create a trial vector,  $X_{t,i}$ .

For binomial crossover:

    {  $p = \text{random number}$

$j_{rand} = \text{int}(\text{rand}[0,1] * D) + 1$

      for  $n = 1$  to  $D$

        { if ( $p < CR$  or  $n = j_{rand}$ )

$X_{t,i} = X_{a,i} + F (X_{b,i} - X_{c,i})$

```

        } else Xt,i = Xi,j
    }

```

Perform selection for each target vector,  $X_i$ , by comparing its function value with that of the trial vector,  $X_{t,i}$ . If  $X_{t,i}$  dominates  $X_i$ , then select  $X_{t,i}$  otherwise select  $X_i$  for the next generation population.

```

        If ( $X_{t,i}$  dominates  $X_i$ )
            Put  $X_{t,i}$  into next generation population
        else
            Put  $X_i$  into next generation population
    } /** End of first for loop***/
} /** End of while loop***/

```

Evaluate the objective functions for each vector.

for  $i = 1$  to NP

$C_{ij} = \text{funct}_j()$ .  $j = 1, \dots, \text{no of objectives}$

Remove all the dominated solutions using any one of the approaches proposed by Deb, (2001). In the present study, the naïve and slow approach is used.

Print the results (after the stopping criteria is met).

### 7.3.1. Test Problems

The algorithm is tested on the following two test problems (Deb, 2001). The first problem is of one dimension while the other is of two dimensions.

*Schaffer's function*

Minimize  $f(x) = x^2$

Minimize  $g(x) = (x-2)^2$

where  $-1000 < x < 1000$

*Cantilever Design Problem*

A cantilever design problem with two decision variables is considered, i.e., diameter ( $d$ ) and length ( $l$ ). The beam has to carry an end weight load  $P$ . the objectives are minimization of weight ( $f_1$ ) and minimization of end deflection ( $f_2$ ). The first objective will resort to an optimum solution having the smaller dimensions of  $d$  and  $l$ , so that the overall weight of the beam is minimum. Since the dimensions are small, the beam will not be adequately rigid and the end deflection of the beam

will be large. On the other hand, if the beam is minimized for end deflection, the dimensions of the beam are expected to be large, thereby making the weight of the beam large.

Minimize  $f_1$  and  $f_2$

$$\text{where } f_1(d, l) = \frac{\rho d^2 l}{4}, \text{ and } f_2(d, l) = \delta = \frac{64Pl^3}{3E\pi d^4}$$

$$\text{Subject to } \sigma_{\max} \leq S_y,$$

$$\delta \leq \delta_{\max}$$

$\sigma_{\max}$  is calculated using  $\sigma_{\max} = \frac{32Pl}{\pi d^3}$  and the following parameters are used  $\rho =$

$7800\text{kg/m}^3$ ,  $P = 1\text{kN}$ ,  $E = 207\text{GPa}$ ,  $S_y = 300\text{Mpa}$ ,  $\delta_{\max} = 5\text{mm}$ ;

### 7.3.2. Results and Discussion

#### 7.3.2.1. Schaffer's function

Various experiments have been carried out in order to test the proposed algorithm by studying the effect of *Max\_gen* and *F* & *CR* for Schaffer's function.

##### 7.3.2.1.1. Effect of *Max\_gen*

The Pareto optimal front obtained using NSDE algorithm is shown in Fig. 7.1. The key parameters used are  $NP = 40$ ,  $CR = 0.5$ ,  $F = 0.2$ , and maximum number of generations (*Max\_gen*) = 300 for a seed value of 10. It can be seen that maximum value of  $f$  is 4.0 and maximum value of  $g$  is 4.0. All other values of  $f$  and  $g$  will lie within these values for Pareto optimal solution. Fig. 7.2 shows the effect of *Max\_gen* on the number of solutions in final Pareto set (*NPS*) taking  $NP = 100$ . It is clear that with increase in *Max\_gen*, *NPS* also increases and after *Max\_gen* = 200, there is no significant increase in *NPS*. Therefore, an appropriate value of *Max\_gen* seems to be about 200 giving  $NPS = 99\%$  of initial population, i.e.,  $NP$ . When ten runs with

different seed values are carried out, the value of *NPS* was found to vary from 95 to 100% with an average of 98%.

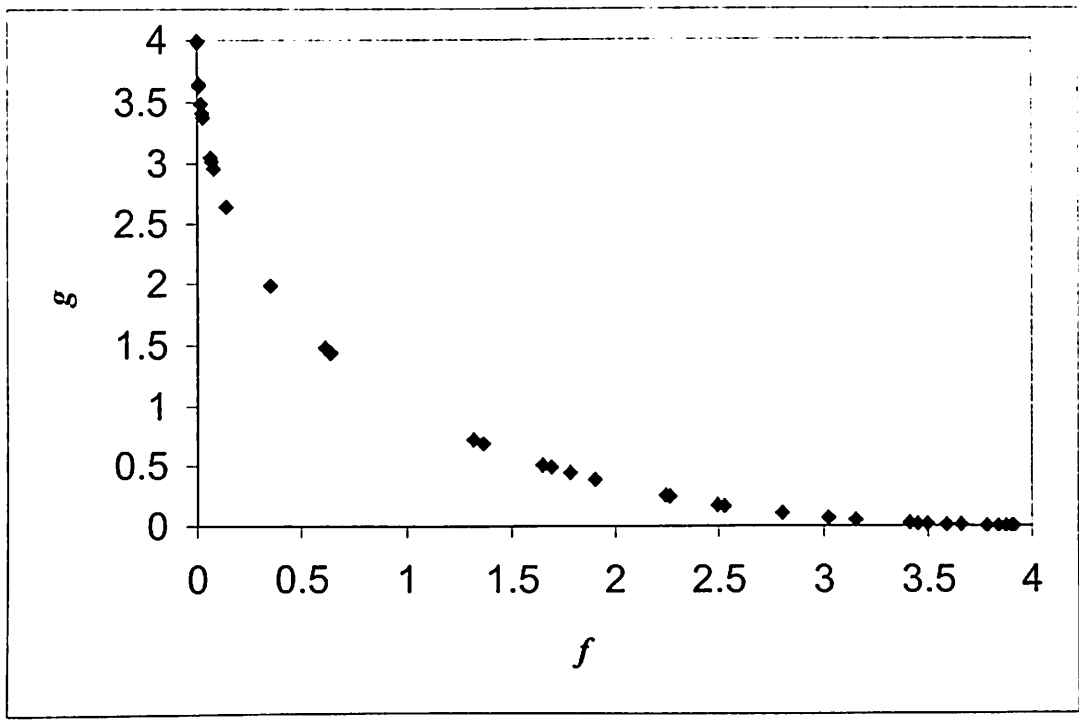


Fig. 7.1. Pareto optimal front for Schaffer's function ( $NP = 40$ )

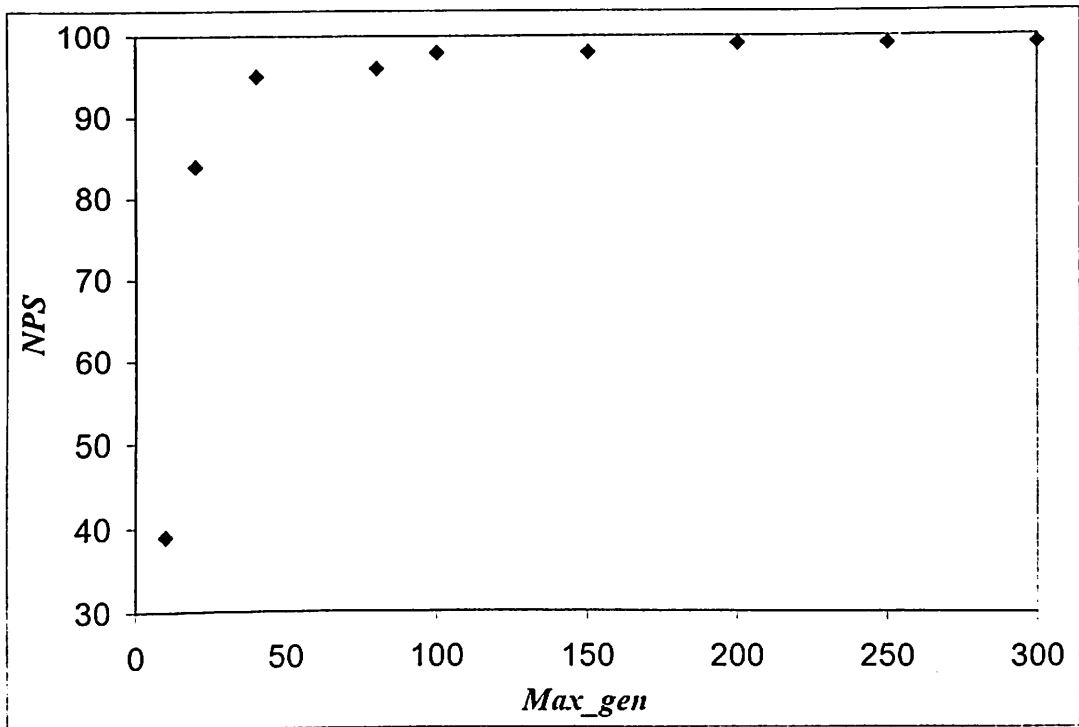


Fig. 7.2. Effect of *Max\_gen* on *NPS*

When an execution is done with  $NP = 400$ ,  $Max\_gen = 300$ , and other parameters same as mentioned earlier,  $NPS$  is found to be 379 (Fig. 7.3) and at  $Max\_gen = 200$ ,  $NPS$  is found to be 375. This further establishes the fact that an appropriate value of  $Max\_gen$  is 200 giving  $NPS$  about 94 to 100% of  $NP$ . However, an appropriate value of  $Max\_gen$  may vary from problem to problem. Higher population size leads to a better Pareto front in terms of number of choices for the optimal solutions (Fig. 7.3).

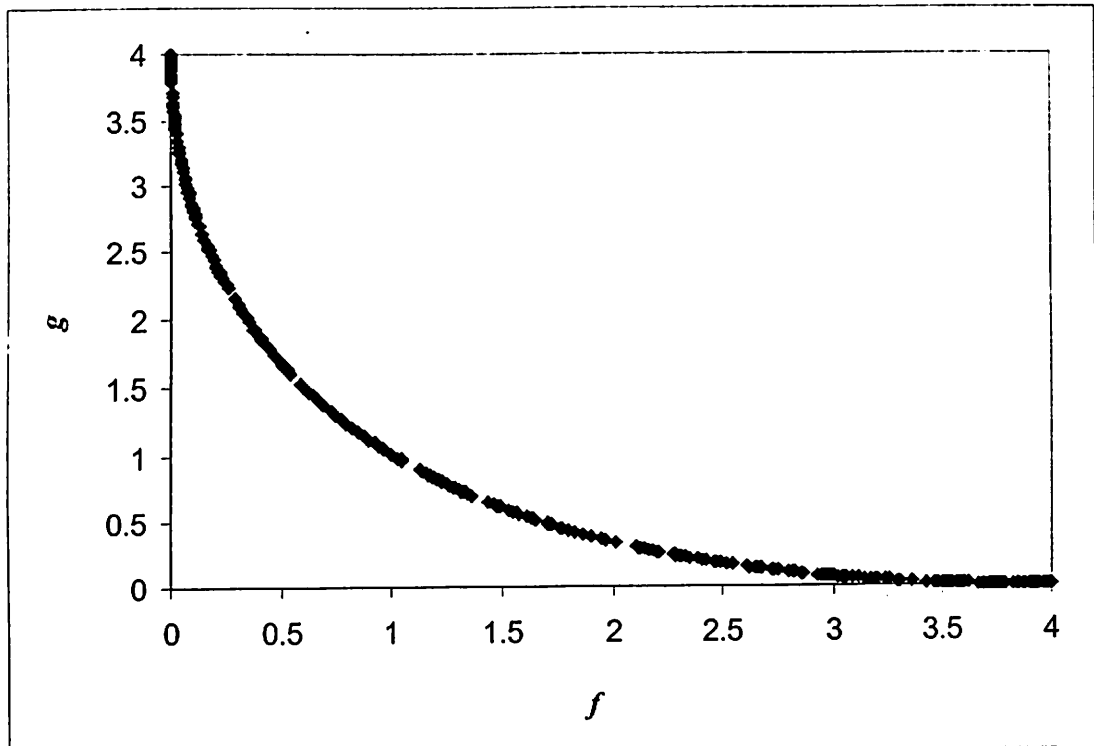
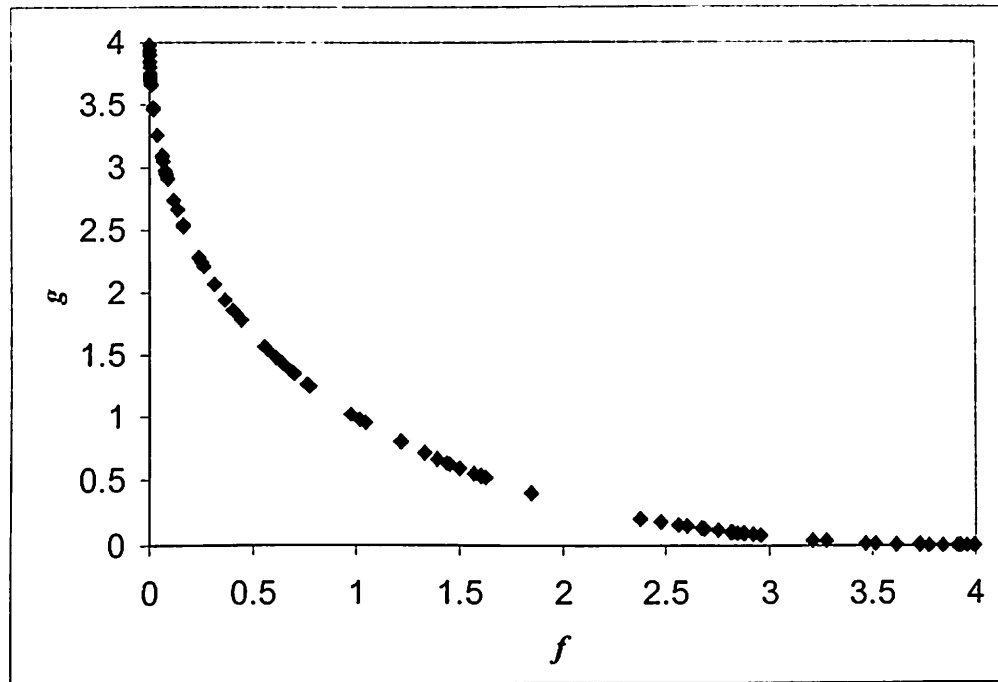


Fig. 7.3. Pareto optimal front for Schaffer's function ( $NP = 400$ )

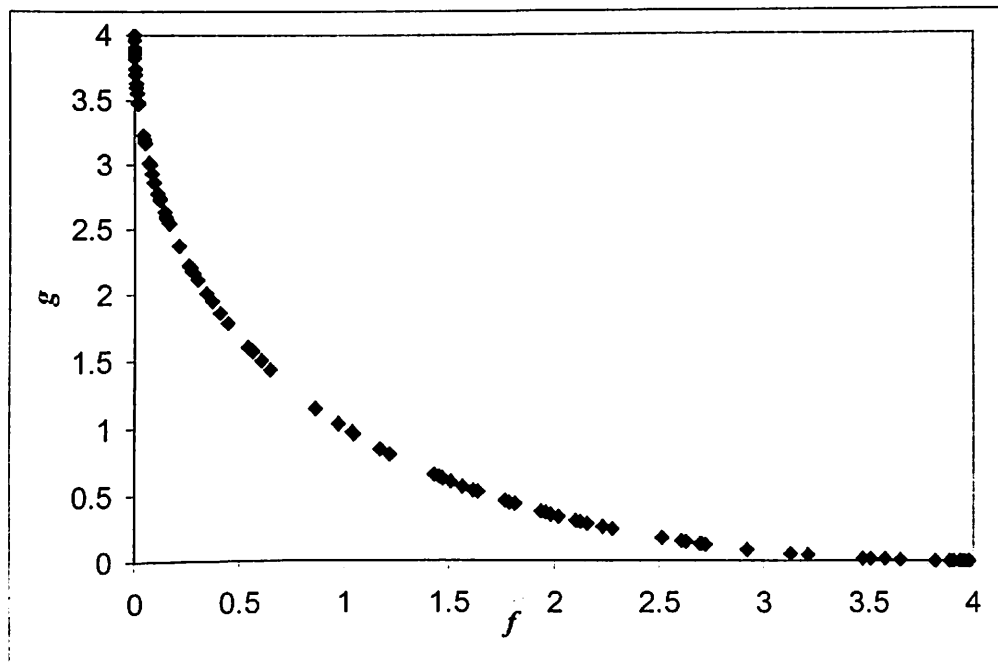
#### 7.3.2.1.2. Effect of $CR$ and $F$

The effect of  $CR$  and  $F$  is studied using the following settings:  $NP = 100$ ,  $Max\_gen = 300$ ,  $CR = 0.5$  if  $F$  is varied, and  $F = 0.5$  if  $CR$  is varied. Fig. 7.4 shows the Pareto front when  $F$  is fixed at 0.5 and  $CR$  is varied in steps of 0.1. It is observed that  $CR$  has no effect on  $NPS$  and pareto front. In all the experiments with  $CR$  varying from 0.1 to 1.0,  $NPS$  is found to be 99% of  $NP$  and exactly same Pareto front is obtained (Fig. 7.4a). However for a different seed value, the distribution of solutions on the Pareto front is different (Fig. 7.4b) but  $NPS$  is nearly same (98). Also, it is

found that  $F$  does have little effect on  $NPS$  (95 to 99) but the distribution (spread) of solutions on Pareto front is different for different values of  $F$  (Fig. 7.5a, 7.5b) for same seed value. It is important to note that spread is different not only for different values of  $F$  but also for different seed values.

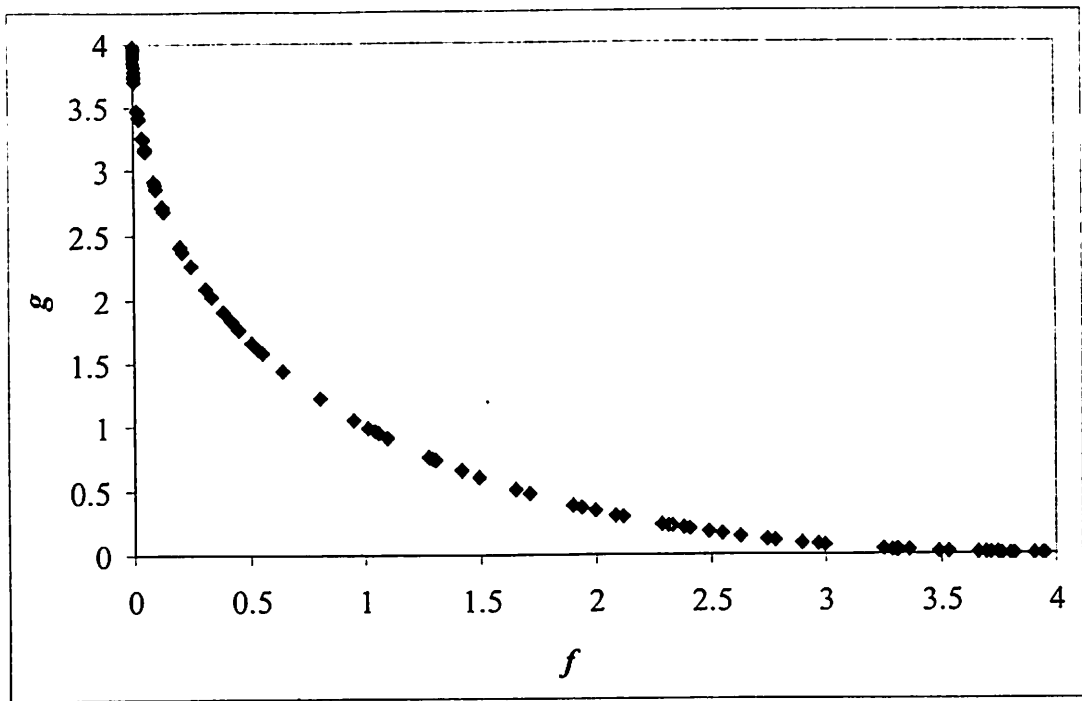


(a) Seed = 10.

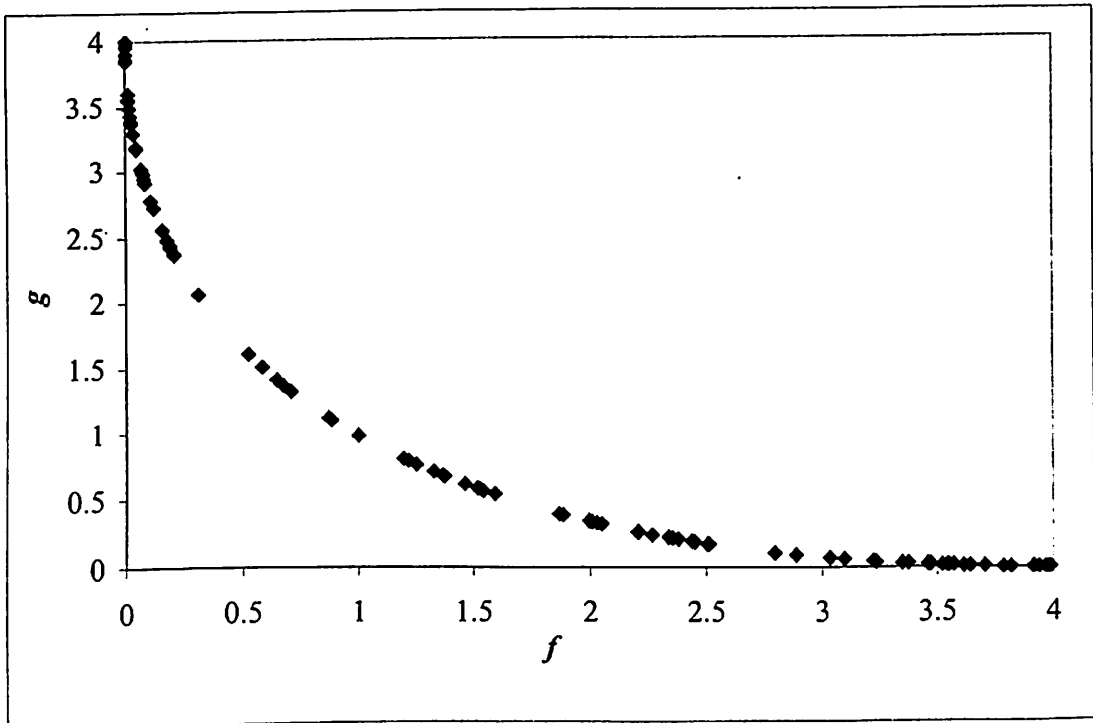


(b) Seed = 23.

Fig. 7.4. Pareto optimal front for Schaffer's function for different seed values

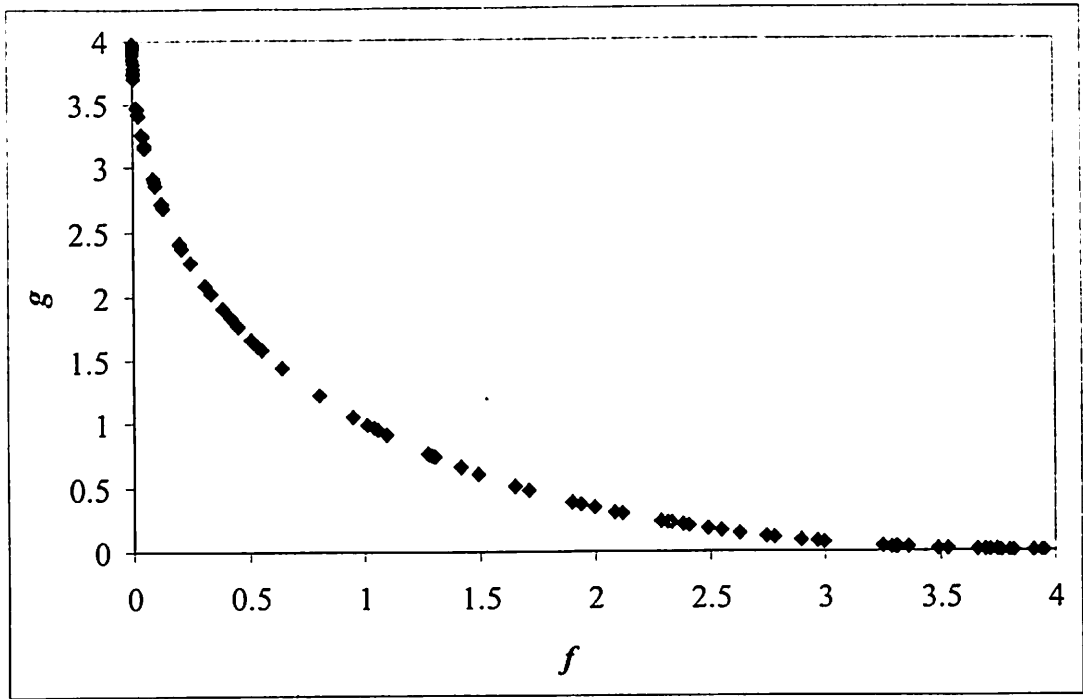


(a) Seed = 10,  $F = 1.2$

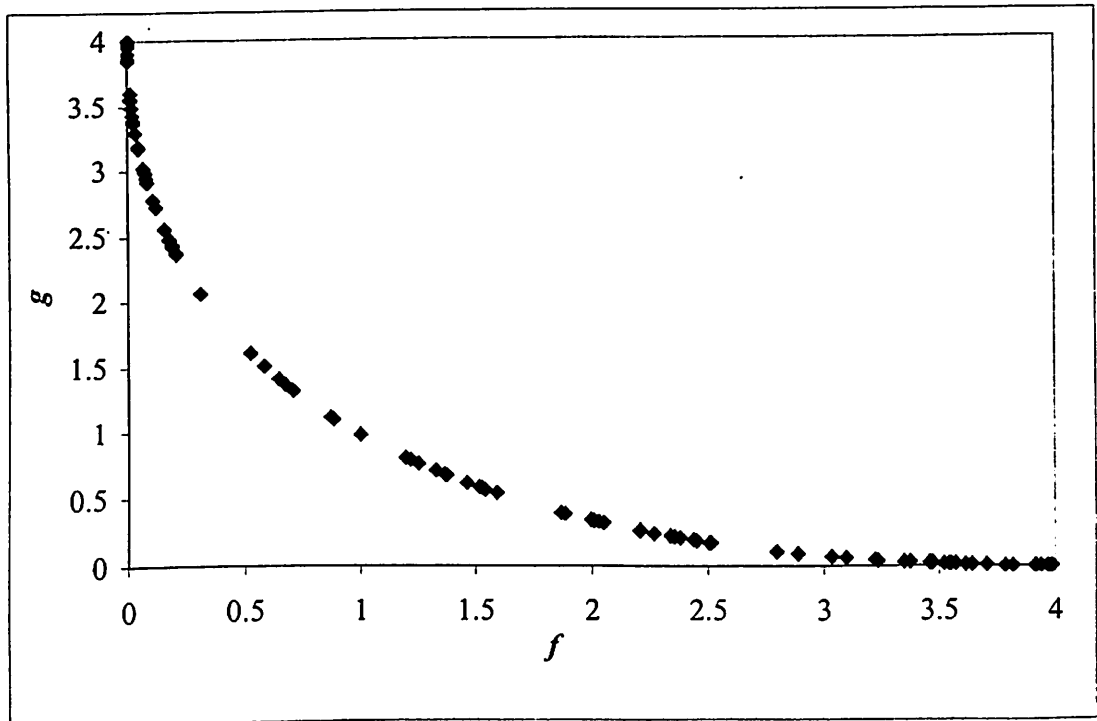


(b) Seed = 10,  $F = 0.2$ .

Fig. 7.5. Pareto optimal front for Schaffer's function for different  $F$  values



(a) Seed = 10,  $F = 1.2$



(b) Seed = 10,  $F = 0.2$ .

Fig. 7.5. Pareto optimal front for Schaffer's function for different  $F$  values



### 7.3.2.2. Cantilever design problem

#### 7.3.2.2.1. Effect of $Max\_gen$

The Pareto optimal front is shown in Fig. 7.6 for the following setting of parameters: population size ( $NP$ ) = 100,  $CR$  = 0.5,  $F$  = 0.5, seed = 11, and  $Max\_gen$  = 500.

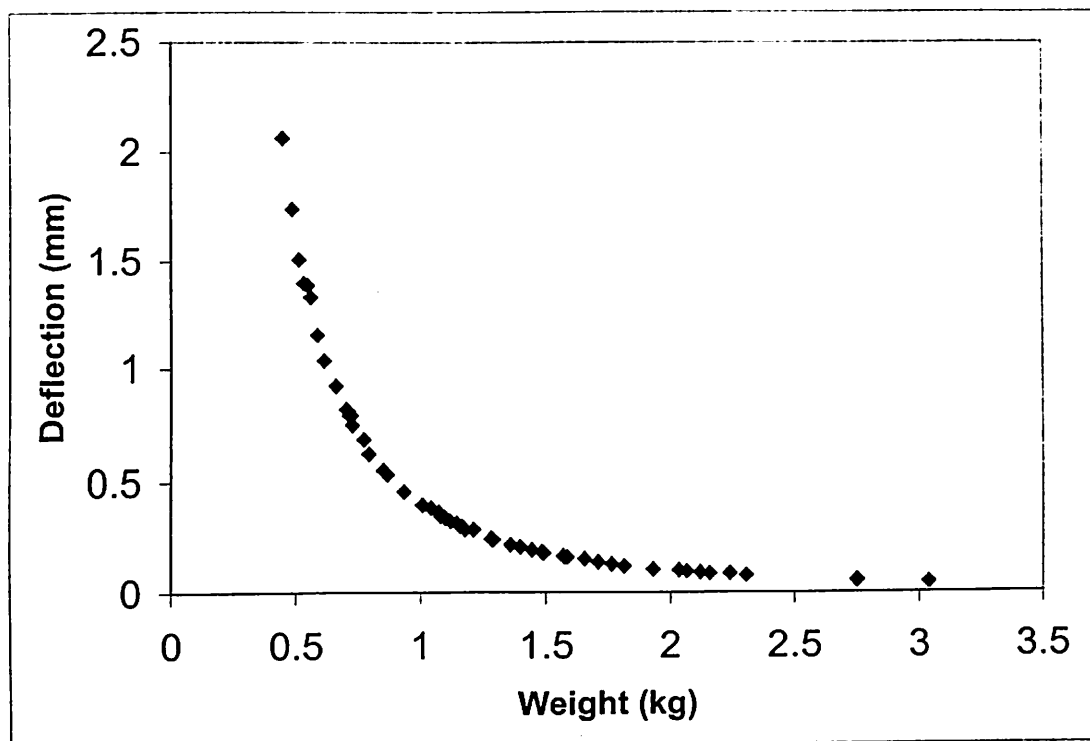


Fig. 7.6. Pareto optimal front for cantilever design problem

It can be seen that the maximum value of weight (kg) is 3.04 and the maximum value of Deflection (mm) is 2.06 as compared to the literature values of 3.06 & 2.04 respectively. All other values of weight and deflection will lie within these values of Pareto optimal solution. Fig. 7.7 shows the effect of  $Max\_gen$  on the number of solutions in final Pareto set ( $NPS$ ). It is clear that with increase in  $Max\_gen$ ,  $NPS$  also increases and after  $Max\_gen$  = 300, there is no significant increase in  $NPS$  but a good shape of Pareto optimal front is found after 500 generations. Therefore, an appropriate value of  $Max\_gen$  seems to be about 500 giving  $NPS$  = 58% of  $NP$ . When ten runs with different seed values are carried out, the value of  $NPS$  was found to vary from 50

to 58% with an average of 54%. When an experiment is done with  $NP = 400$  and keeping other parameters same as mentioned earlier,  $NPS$  is found to be 34.5% as against 54% for  $NP = 100$ . At  $Max\_gen = 700$  and 1000,  $NPS$  is found to be 39% and 45% respectively. This indicates that increase in  $NP$  does not lead to a corresponding increase in  $NPS$  if number of  $Max\_gen$  is held constant. This is not similar to that found for Schaffer's function. Higher population size needs higher number of generations to get a better Pareto front in terms of both quantity ( $NPS$ ) and quality (spread & shape, i.e., global Pareto front).

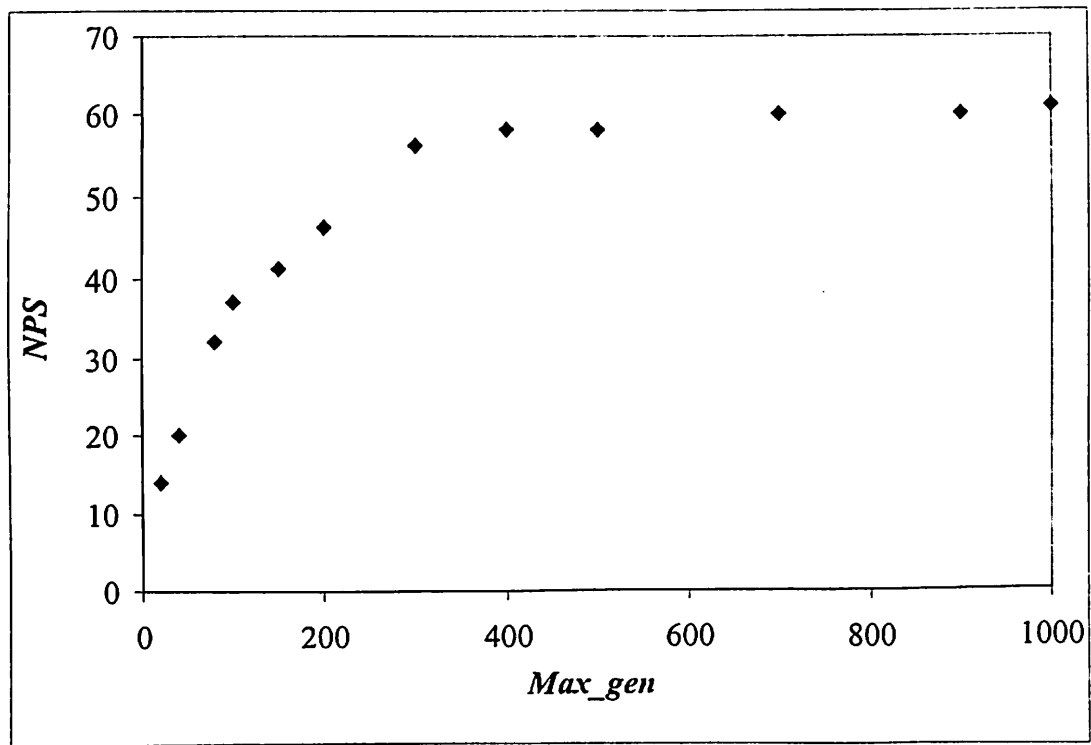
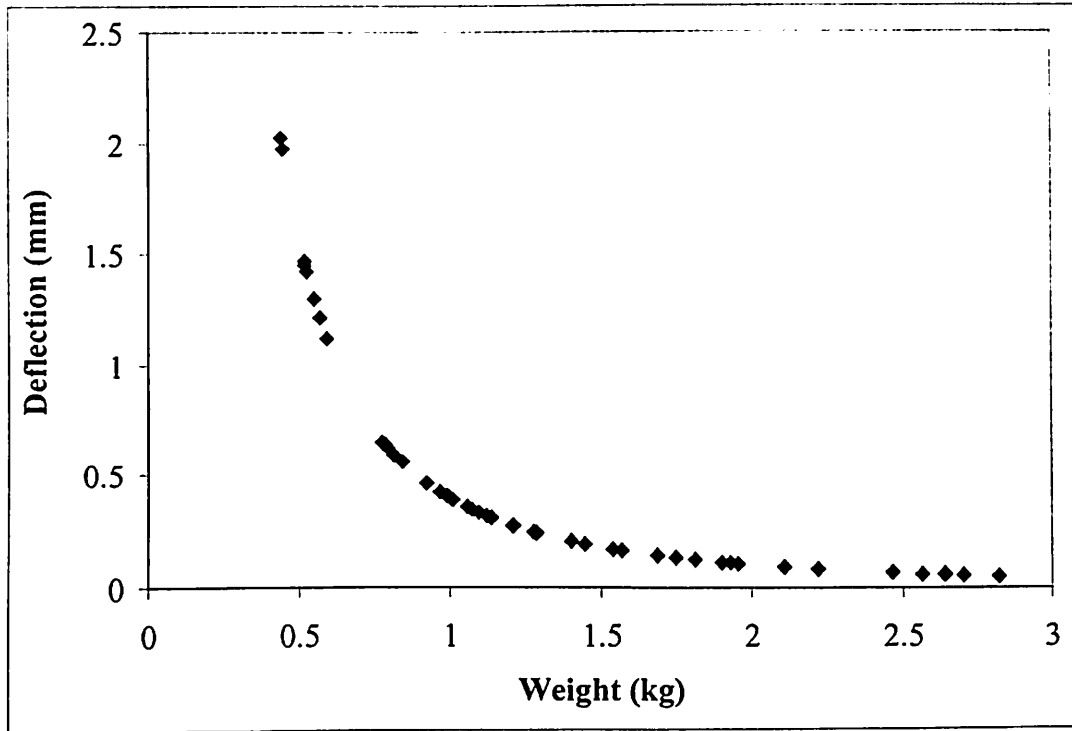


Fig. 7.7. Effect of  $Max\_gen$  on  $NPS$  (cantilever design problem)

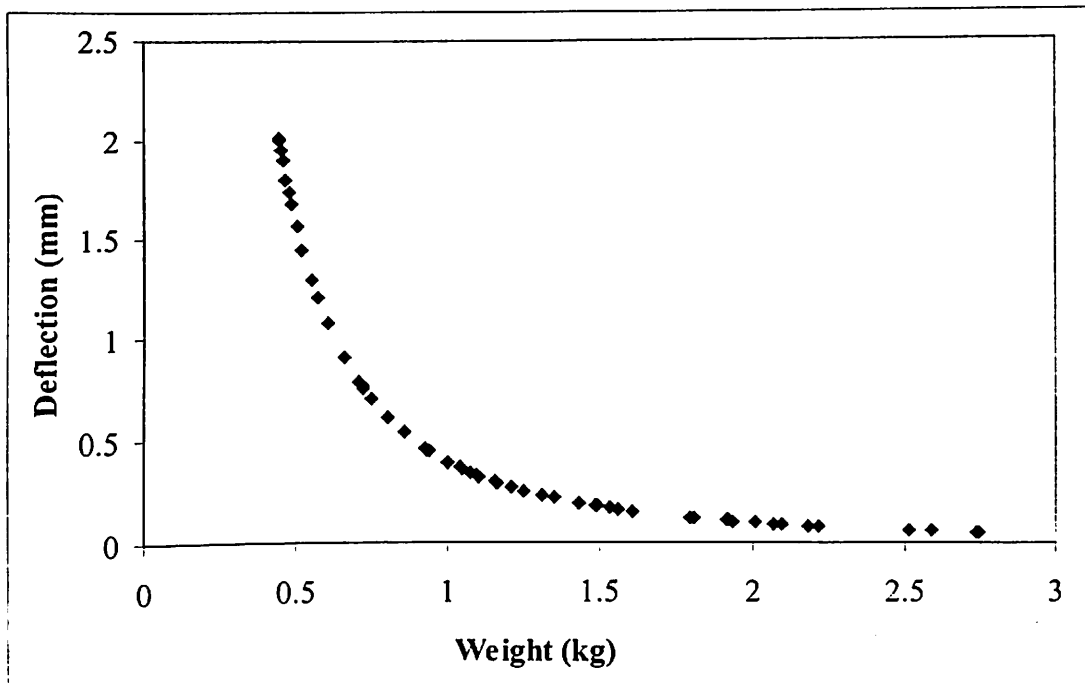
#### 7.3.2.2.2. Effect of $CR$ & $F$

First the effect of  $F$  is studied using  $CR = 0.5$ ,  $NP = 100$ ,  $Max\_gen = 500$ , seed = 10. Fig. 7.8 shows the Pareto optimal solution obtained for two different values of  $F$  (0.1 and 0.5). It is clear that the value of  $F$  not only affects the  $NPS$  but also the distribution (spread) of Pareto optimal solutions on Pareto front. It is found that the values of  $NPS$  are 47 and 58 for  $F = 0.1$  and 0.5 respectively.  $NPS$  is found to vary

from 47 to 58% as shown in Fig. 7.9. As is evident from Fig. 7.9,  $F = 0.5$  seems to be good giving highest  $NPS$  and good distribution of solutions on Pareto front (Fig. 7.8b).



(a).  $F = 0.1$



(b).  $F = 0.5$

Fig. 7.8. Pareto optimal front for different  $F$  values (cantilever design problem)

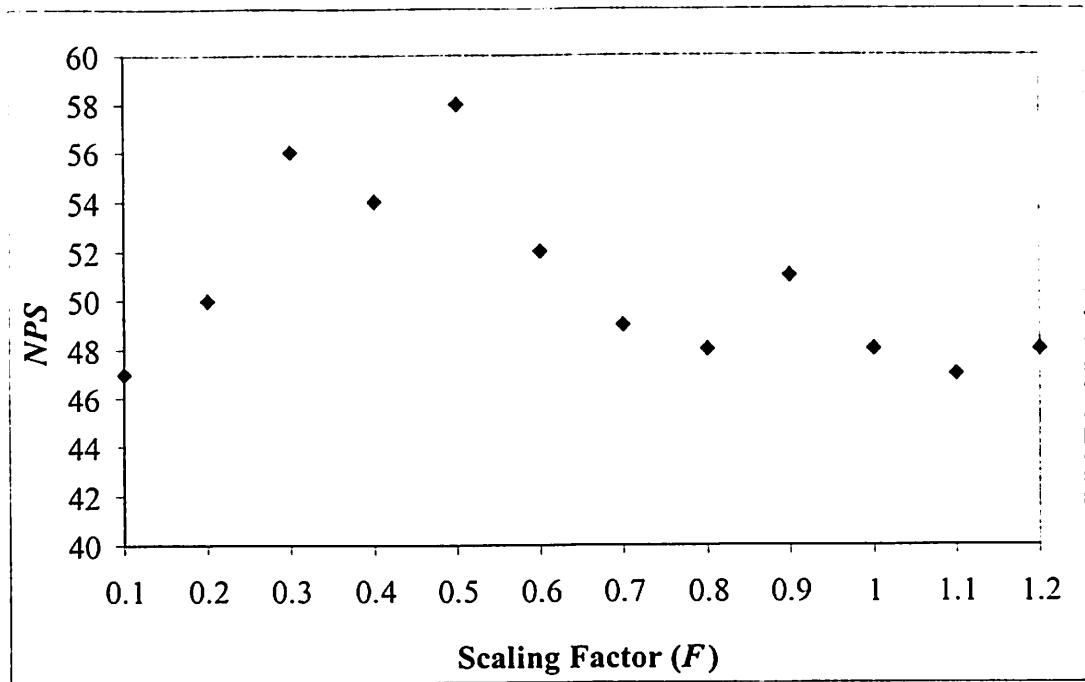


Fig. 7.9. Effect of scaling factor ( $F$ ) on  $NPS$  (cantilever design problem)

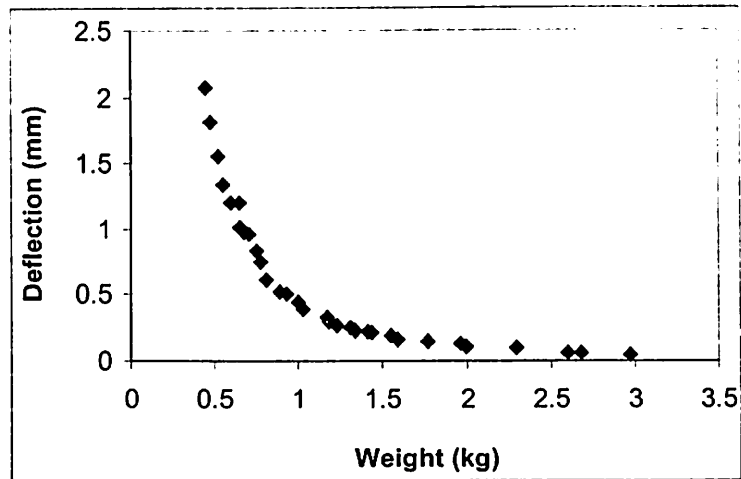
The change in maximum and minimum values of objective functions with  $F$  is shown in Table-7.1. This gives information about how sensitive the extreme values of objective functions are as value of  $F$  changes. And the variation is significant in the values of Weight (max) & Deflection (max) for  $F = 0.5$  and  $0.8$ .

Table-7.1. Variation of objective function values (min & max) with  $F$

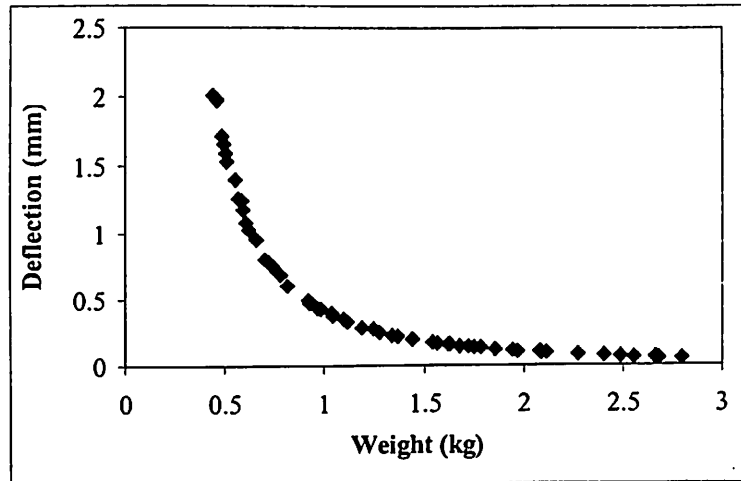
$F$	Weight (kg)		Deflection (mm)	
	Min	Max	Min	Max
0.1	0.4412	2.8218	0.0500	2.0285
0.5	0.4426	2.7471	0.0526	2.0169
0.8	0.4668	2.9966	0.0499	1.8470
1.2	0.4672	2.9657	0.0490	1.9456

To study the effect of  $CR$ , the parameter setting is as follows:  $F = 0.5$ ,  $NP = 100$ ,  $Max\_gen = 500$ ,  $seed = 10$ . The  $CR$  is changed in steps of 0.1 from 0.1 to 1.0. Fig. 7.10a, 7.10b, and 7.10c show the Pareto optimal solutions for  $CR = 0.1$ , 0.5, and 1.0 respectively. Fig. 7.11 shows the variation of  $NPS$  with  $CR$ . It is clear from Fig. 7.10 that higher value of  $CR$  ( $\cong 1.0$ ) results in good shape, i.e., global Pareto front. As shown in Fig. 7.10a, some of the solutions are not lying on the Pareto front for lower

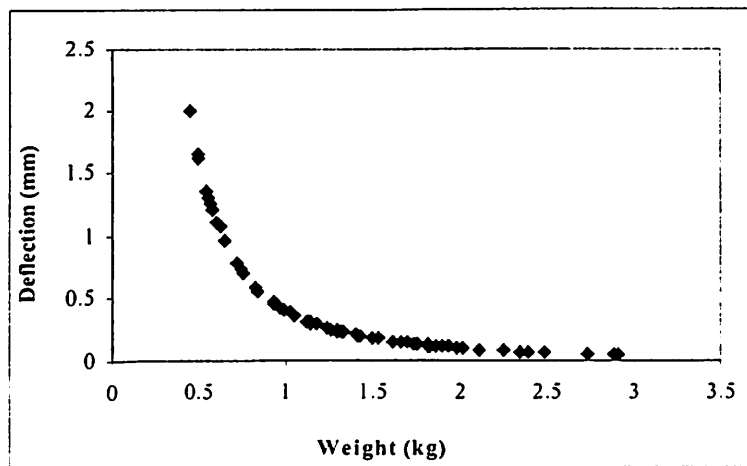
value of  $CR$  i.e. 0.1 & 0.5. Also,  $NPS$  increases with  $CR$  (Fig. 7.11). This indicates that for a given  $Max\_gen$ , a higher value of  $CR \cong 1.0$  works better.



(a).  $CR = 0.1$



(b).  $CR = 0.5$



(c).  $CR = 1.0$ .

Fig. 7.10. Pareto optimal fronts for different  $CR$  values (cantilever design problem)

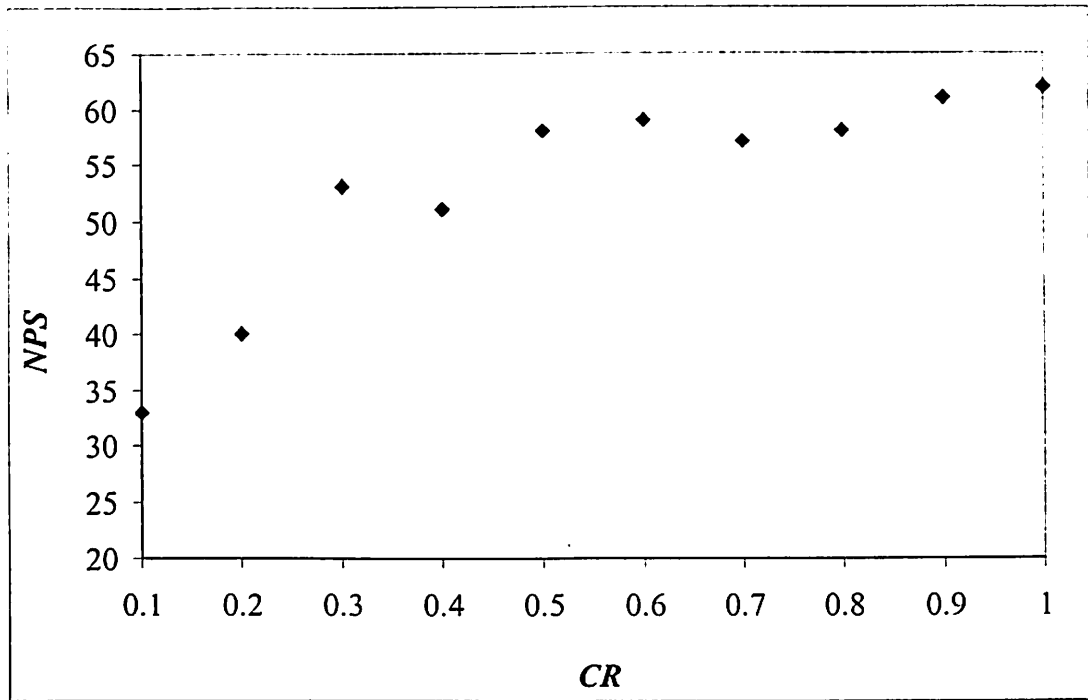


Fig. 7.11. Effect of  $CR$  on  $NPS$  (cantilever design problem)

The change in maximum and minimum values of objective functions with  $F$  is shown in Table-7.2. This is clear from Table-7.2 that extreme values of the objective functions change with  $CR$ . However the variation is small as compared to effect of  $F$  (Table-7.1).

Table-7.2. Variation of objective function values (min & max) with  $CR$

$CR$	Weight (kg)		Deflection (mm)	
	Min	Max	Min	Max
0.1	0.4465	2.9703	0.0447	2.0723
0.5	0.4430	2.7960	0.0531	2.0095
1.0	0.4451	2.9103	0.0469	1.9982

### 7.3.3. Conclusions

Two problems, one standard test problem and cantilever design problem are solved using proposed NSDE algorithm. The results indicate that NSDE is able to locate the global Pareto front for two test problems studied. The effects of various parameters of NSDE, i.e.,  $Max\_gen$ ,  $CR$ , and  $F$  are discussed & analyzed. It is found that for both the test problems an increase in  $Max\_gen$  increases the  $NPS$  up to a

certain value, and after that there is no significance increase in *NPS*. This certain value is problem dependent and is found to be different for the two problems studied. For Schaffer's function it is about 200, while for cantilever design problem it is about 500. The effect of *CR* is significant in case of cantilever design problem while it does not affect the Schaffer's function. A high value of *CR* ( $\cong 1$ ) is found suitable for both the problems. It is important to note that *F* not only affects the *NPS* but also the distribution of solutions in Pareto front for both the problems. A value of *F* ( $= 0.5$ ) is found suitable for both the problems.

#### **7.4. Modified Non-Dominated Sorting Differential Evolution (MNSDE)**

As we have seen already in Chapter-3 and Chapter-4 that MDE took less computational time due to the use of single array of population. In this chapter, MDE is extended for solving multi-objective optimization problems and the extended algorithm is called as MNSDE (Modified Non-dominated Sorting Differential Evolution). MNSDE is similar to NSDE except for the selection criterion. Also, MNSDE maintains only one set of population as against two sets in NSDE. The selection criterion used in MNSDE is different from that of NSDE and is as follows:

After mutation & crossover the trial solution is generated. Selection is made between this trial solution and target solution. If trial solution dominates the target solution, then the target solution is replaced by the trial solution in the population of current generation itself otherwise the target solution is kept as it is. The remaining procedure is same as that of NSDE. The use of single array of population in MNSDE as against two in NSDE may lead to reduction in memory and computational efforts required as is found for MDE.

## 7.4.1. Results & Discussion

### 7.4.1.1. Schaffer's function

Many experiments have been carried out in order to test the proposed algorithm by studying the effect of  $Max\_gen$  and  $F$  &  $CR$  for Schaffer's function. Fig. 7.12 shows the Pareto front using the two techniques, i.e., NSDE, and MNSDE. Parameters used are same in both the techniques except for the  $Max\_gen$ . For MNSDE,  $Max\_gen$  is 100, while it is 200 for NSDE. This is done in order to examine the effect of use of single array instead of double in NSDE. It is clear that both MNSDE & NSDE are able locate global Pareto optimal front but with different spread. It is to be noted that even though the  $Max\_gen$  used in MNSDE is half of that used in NSDE, it is able to locate the global Pareto front.

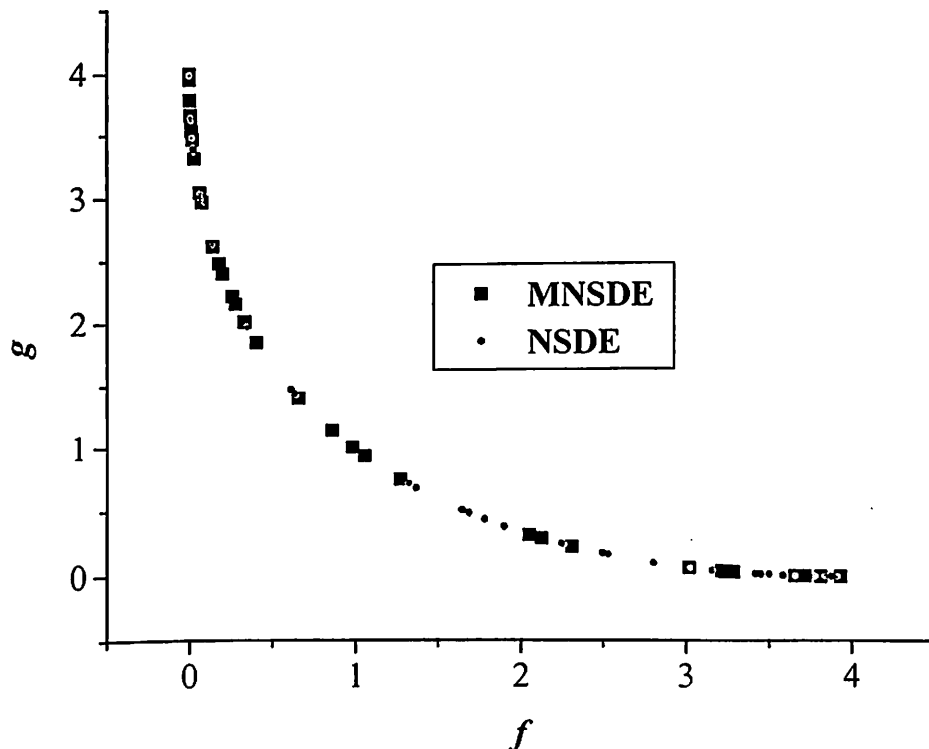


Fig. 7.12. Pareto front using MNSDE & NSDE (Schaffer's function)



#### 7.4.1.1.1. Effect of $Max\_gen$ on MNSDE & its Comparison with NSDE

Fig. 7.13 shows the effect of  $Max\_gen$  on MNSDE and its comparison with that of NSDE. From Fig. 7.13, it is clear that effect of  $Max\_gen$  is same on the two techniques. Key parameters used are  $F = 0.5$ ,  $CR = 0.5$ ,  $NP = 100$ . However, there is no significant change in  $NPS$  after  $Max\_gen = 100$  for MNSDE while same trend is observed for NSDE but after  $Max\_gen = 200$ . Also maximum  $NPS$  obtained using NSDE is 99 as compared to 96 using MNSDE.

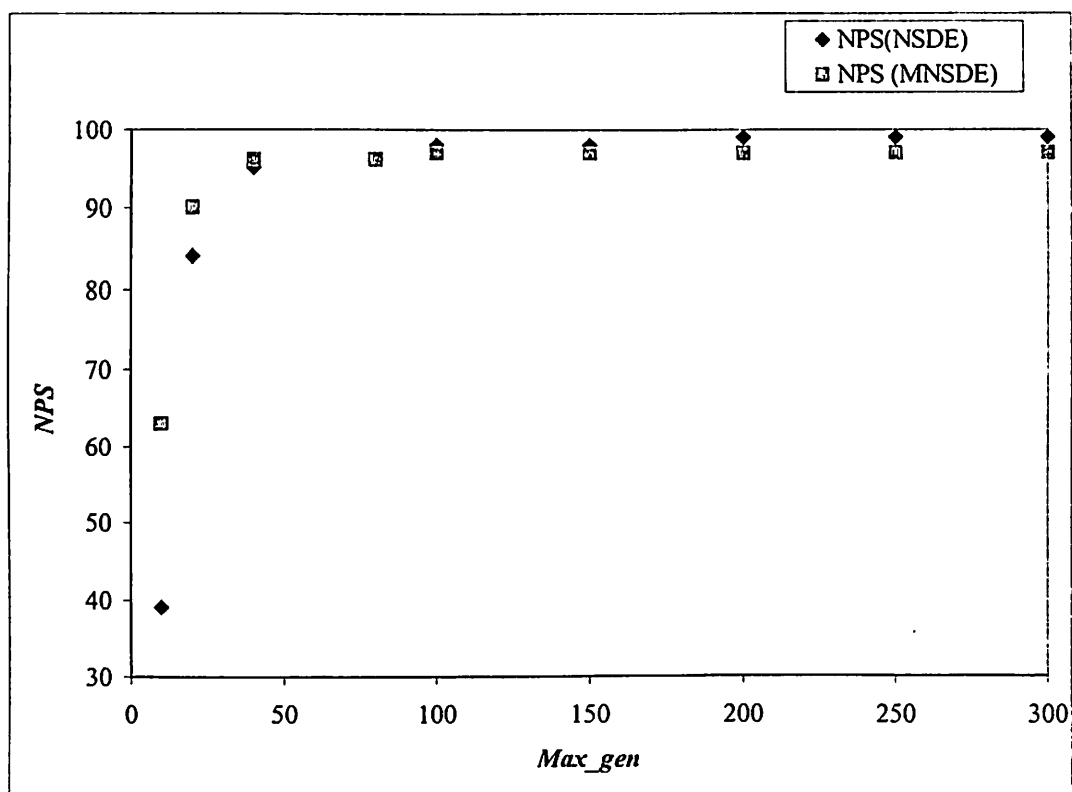


Fig. 7.13. Effect of  $Max\_gen$  on  $NPS$  using MNSDE & NSDE (Schaffer's function)

#### 7.4.1.1.2. Effect of $CR$ on MNSDE & its Comparison with NSDE

Fig. 7.14 shows the effect of  $CR$  on the performance of MNSDE. Fig. 7.15 shows the effect of seed and its comparison with NSDE. Keeping  $F = 0.5$ ,  $NP = 100$ , and seed = 10, the  $CR$  value is changed from 0.1 to 1.0 in steps of 0.1. The parameters used are same for MNSDE & NSDE except  $Max\_gen = 200$  & 300 respectively. The effect of change in  $CR$  value on MNSDE is similar to that found for NSDE, i.e., there

is no effect on *NPS* and Pareto front obtained (Fig.7.14). Also, different seed values give different spread of Pareto front (Fig. 7.15a & 7.15b). However as compared to NSDE, the spread is different.

Table-7.3 shows the effect of seed value of on maximum objective function value. It is observed that for the two seed value maximum objective function values are almost same as was expected. It is also seen that *NPS* is slightly affected (95 & 98) in case of MNSDE but it is nearly same (98 & 99) for NSDE.

**Table-7.3. Comparison of maximum function values for two different seeds**

Seed	<i>NPS</i> (NSDE/MNSDE)	<i>f</i> (max)		<i>g</i> (max)	
		MNSDE	NSDE	MNSDE	NSDE
10	99/98	3.9290	3.9949	3.9972	4.0137
23	98/95	4.0016	4.0086	3.9928	3.9981

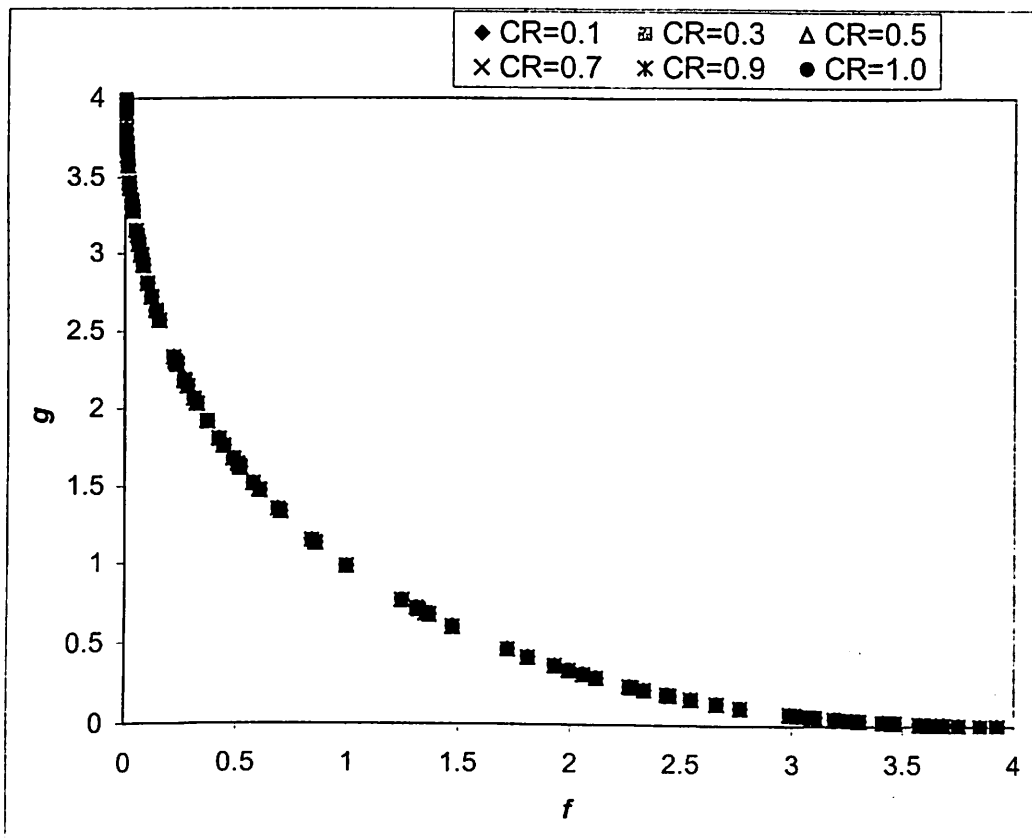
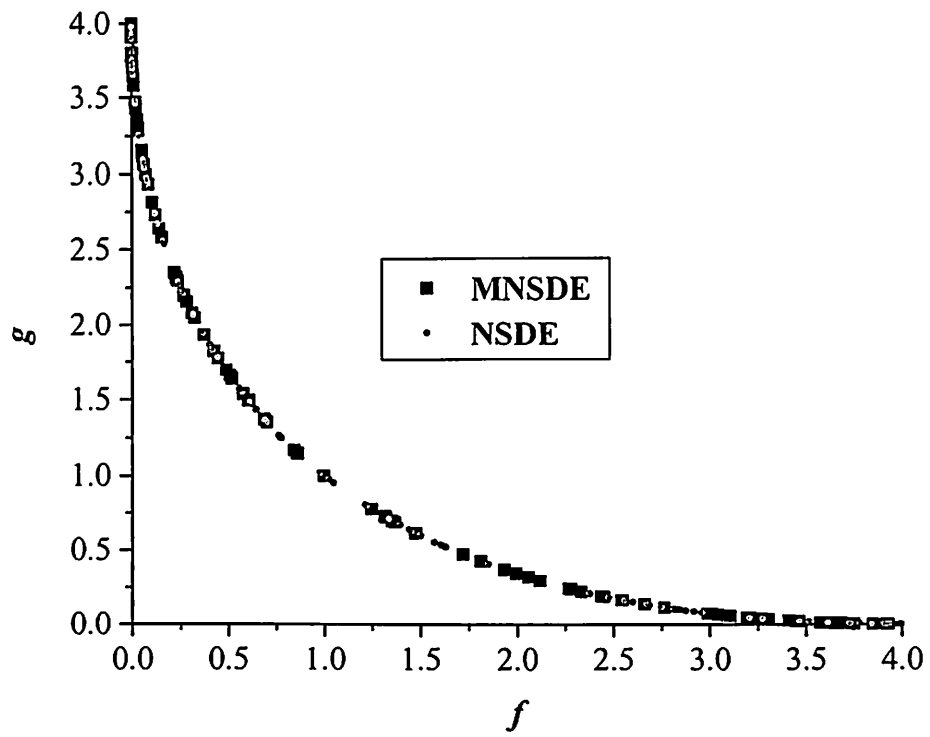
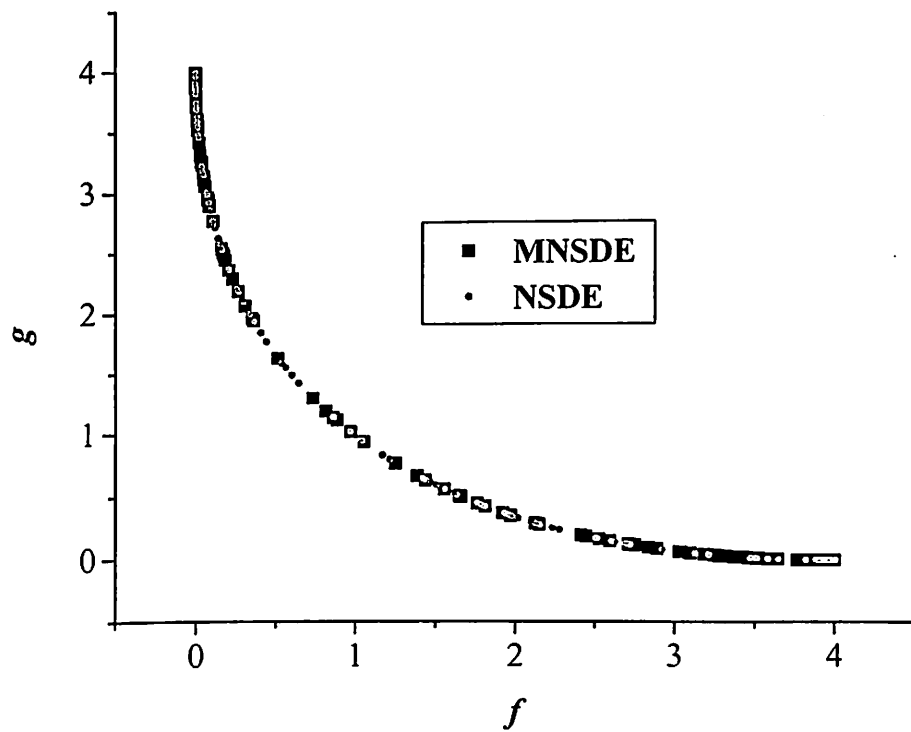


Fig. 7.14. Effect of *CR* using MNSDE (Schaffer's function)



(a). Seed = 10



(b). Seed = 23

Fig. 7.15. Effect of seed for Schaffer's function using MNSDE and NSDE

### 7.4.1.1.3. Effect of $F$ on MNSDE & its Comparison with NSDE

It is found that  $F$  not only affects  $NPS$ , but also the spread as found in NSDE too. The variation of  $NPS$  with  $F$  is shown in Fig. 7.16 for both MNSDE & NSDE. It is clear that the effect of  $F$  on MNSDE is more significant than that on NSDE for the same seed value. In case of MNSDE,  $NPS$  varies from 88 to 99, while for NSDE it varies from 95 to 99. Fig. 7.17 shows the comparison of MNSDE & NSDE for  $F = 0.2$ . Both the techniques are able to find out global Pareto front but spread is different.

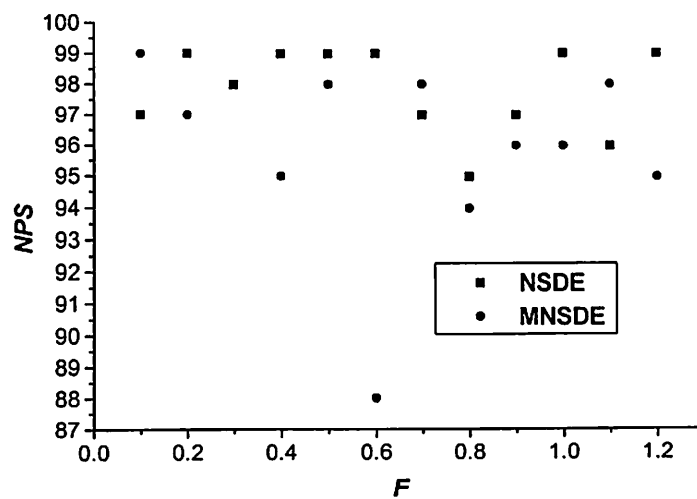


Fig. 7.16. Effect of  $F$  on  $NPS$  for Schaffer's function (MNSDE and NSDE)

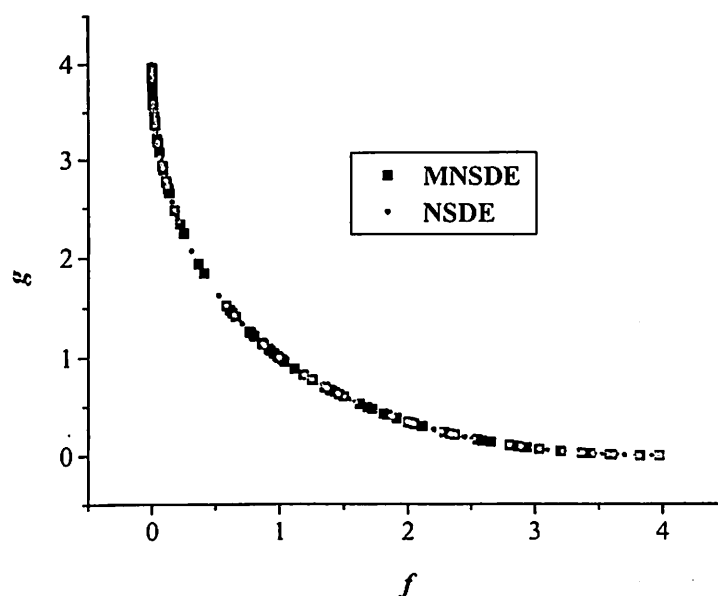


Fig. 7.17. Comparison of MNSDE and NSDE at seed =10 and  $F = 0.2$

#### 7.4.1.2. Cantilever design Problem

Various experiments have been carried out in order to test the proposed algorithm by studying the effect of  $Max\_gen$  and  $F$  &  $CR$  for Schaffer's function.

##### 7.4.1.2.1. Effect of $Max\_gen$ on MNSDE & its Comparison with NSDE

Fig. 7.18 shows the effect of  $Max\_gen$  on the performance of MNSDE and its comparison with that of NSDE. The key parameters used are  $F = 0.5$ ,  $CR = 0.5$ ,  $NP = 100$ . It is seen from Fig. 7.18 that there is not significant difference in  $NPS$  till  $Max\_gen = 500$  for the two techniques. And after  $Max\_gen = 500$ ,  $NPS$  in case of NSDE becomes almost constant with further increase in  $Max\_gen$ . But in the case of MNSDE, it increases even after  $Max\_gen = 500$  and becomes nearly constant after  $Max\_gen = 900$ .  $NPS$  is more in the case of MNSDE than for NSDE for higher values of  $Max\_gen$  ( $>500$ ). This is different from what is observed in the result with Schaffer' function.

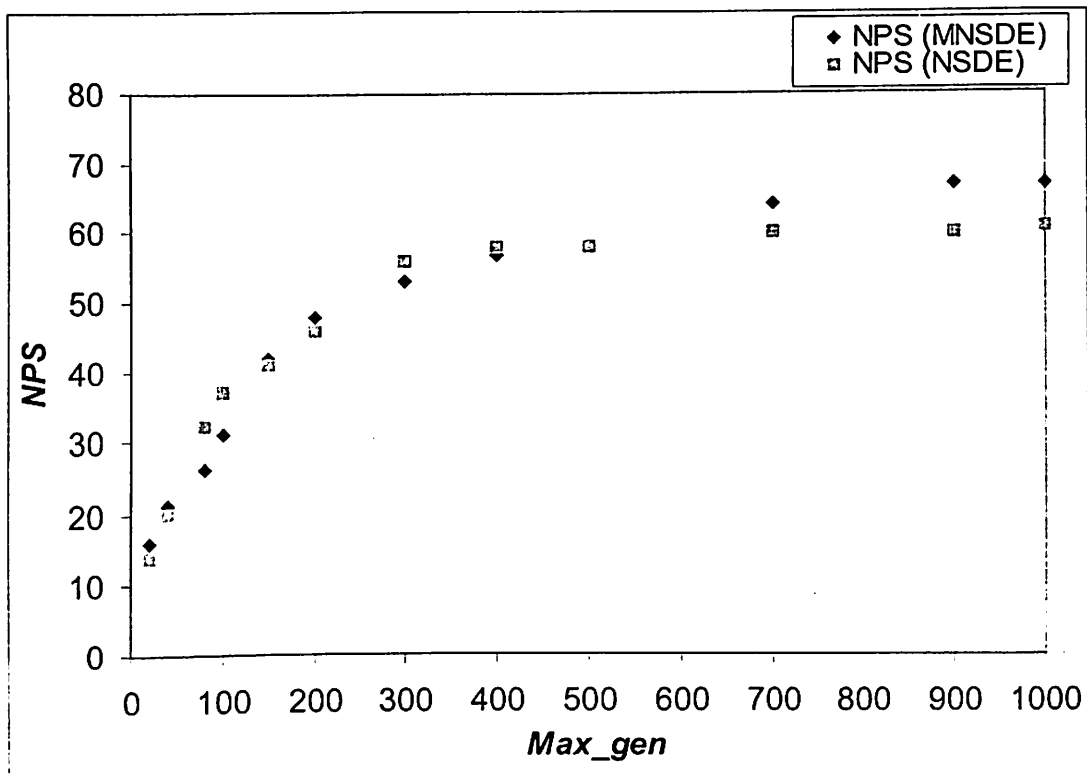


Fig. 7.18. Effect of  $Max\_gen$  on  $NPS$  for cantilever design problem (MNSDE and NSDE)

#### 7.4.1.2.2. Effect of $F$ on MNSDE & its Comparison with NSDE

Fig. 7.19 shows the effect of  $F$  on  $NPS$  using MNSDE and its comparison with that using NSDE. From Fig. 7.19 it is seen that for MNSDE,  $NPS$  is high for lower value of  $F$  while a value of  $F = 0.5$  gives highest  $NPS$  for NSDE. This is in agreement with what is observed in Schaffer's function. It is interesting to note that at  $F = 0.5$ ,  $NPS$  is same for both the techniques. Hence this value can be used for further comparison of Pareto front obtained in the two algorithms. It is found that  $F$  not only affects the  $NPS$  but also the spread of Pareto front. Also, the spread of Pareto front is different for different seed value. Fig. 7.20 shows the Pareto front obtained using the two algorithms for  $F = 0.5$ . Although the spread is different yet both the algorithms are able to locate the global Pareto front.

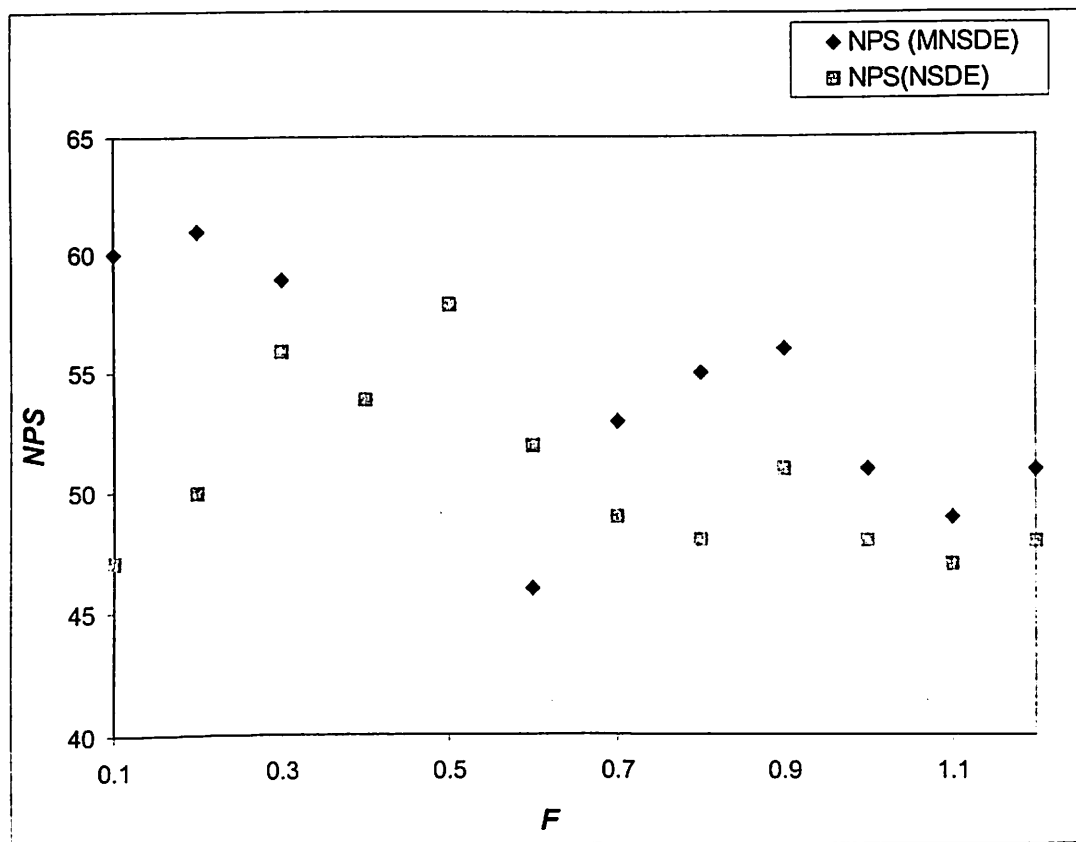


Fig. 7.19. Effect of  $F$  on  $NPS$  for cantilever design Problem using MNSDE and NSDE ( $CR = 0.5$ )

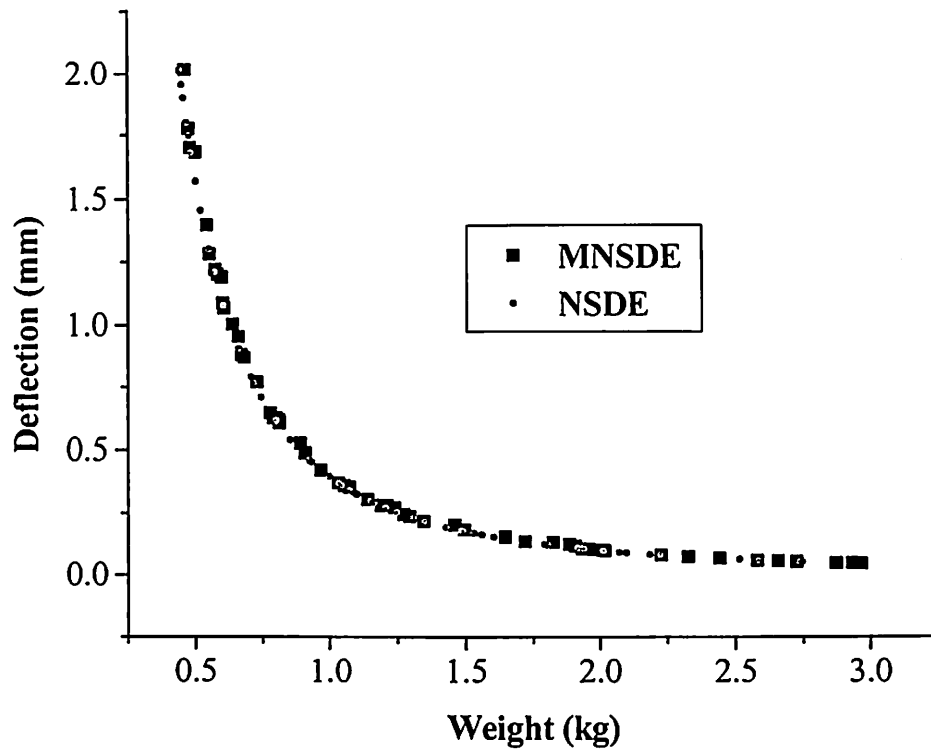


Fig. 7.20. Pareto front for cantilever design problem using MNSDE and NSDE

#### 7.4.1.2.3. Effect of CR on MNSDE & its Comparison with NSDE

The effect of *CR* on *NPS* (for both MNSDE & NSDE) for cantilever design problem is significant as compared to Schaffer's function. The variation of *NPS* with *CR* is shown in Fig. 7.21. Best value of *CR* seems to be 0.7 for MNSDE and 1.0 for NSDE. However, the value of *NPS* is nearly same for  $CR = 0.9$  & 1.0 for both the algorithms.

Fig. 7.22 shows the Pareto fronts for different *CR* values using MNSDE. It is evident that as *CR* value is increased for given *F* & *Max\_gen*, the shape of the Pareto front gets improved. In other words, it is closest to global Pareto front for  $CR \cong 1.0$  (Fig. 7.22c) rather than for lower *CR* values (Fig. 7.22a & 7.22b). This is similar to what is observed with the results using NSDE. The effect of *CR* value on the maximum value of objective function and *NPS* is shown in Table-7.4. Also a comparison is made with the results obtained using NSDE.

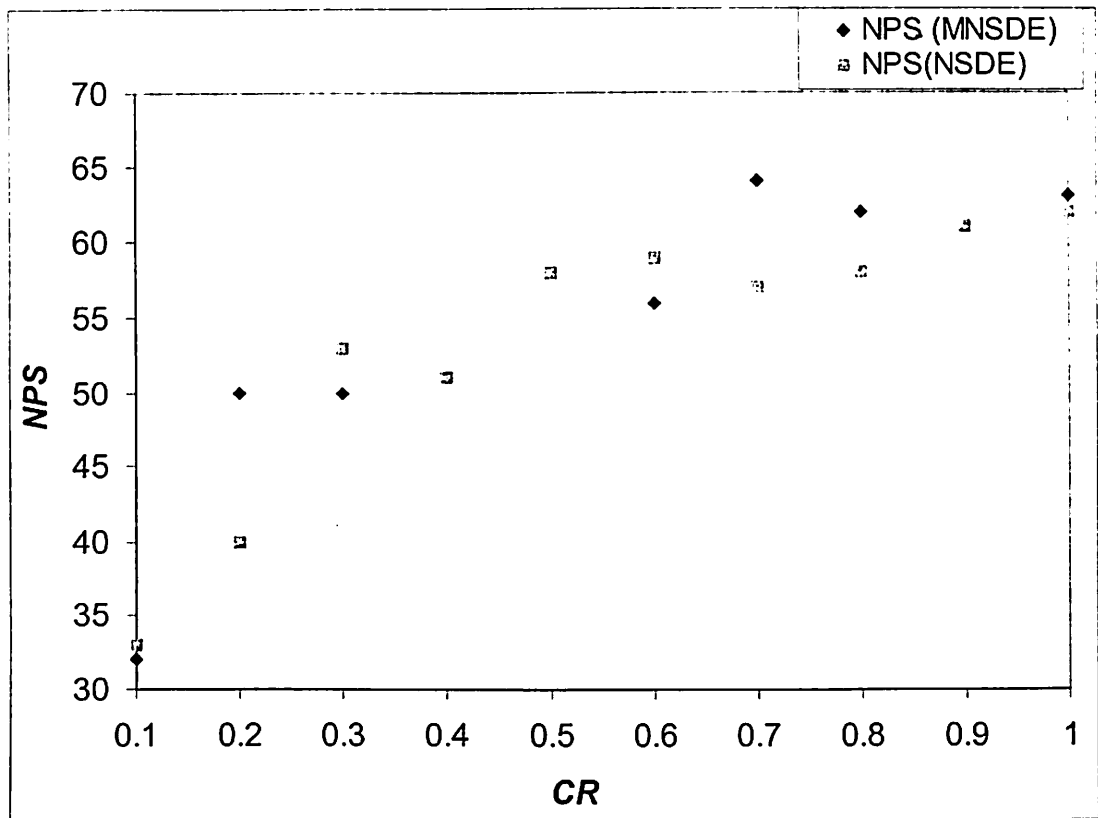


Fig. 7.21. Effect of  $CR$  on  $NPS$  for cantilever design problem (MNSDE and NSDE)

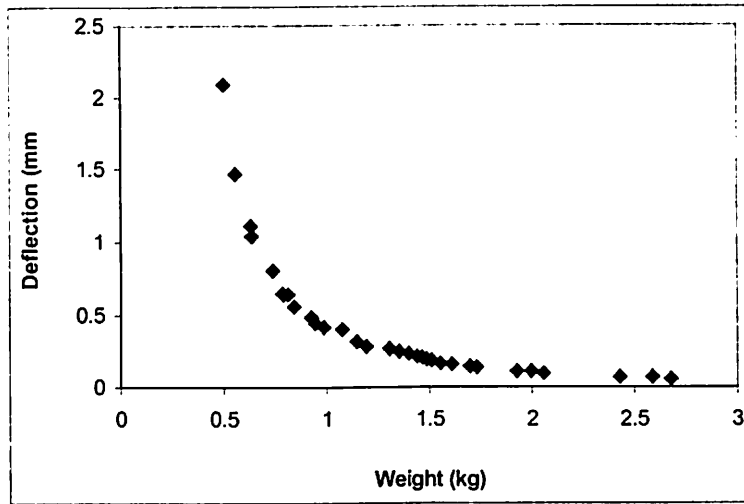
Table-7.4. Effect of  $CR$  on maximum value\* of objective functions

$CR$	$NPS$ (NSDE/MNSDE)	Weight (max)		Deflection (max)	
		MNSDE	NSDE	MNSDE	NSDE
0.1	33/32	2.6789	2.9703	2.0929	2.0723
0.5	58/58	2.9683	2.7960	2.0173	2.0095
1.0	62/63	2.9591	2.9103	1.8434	1.9982

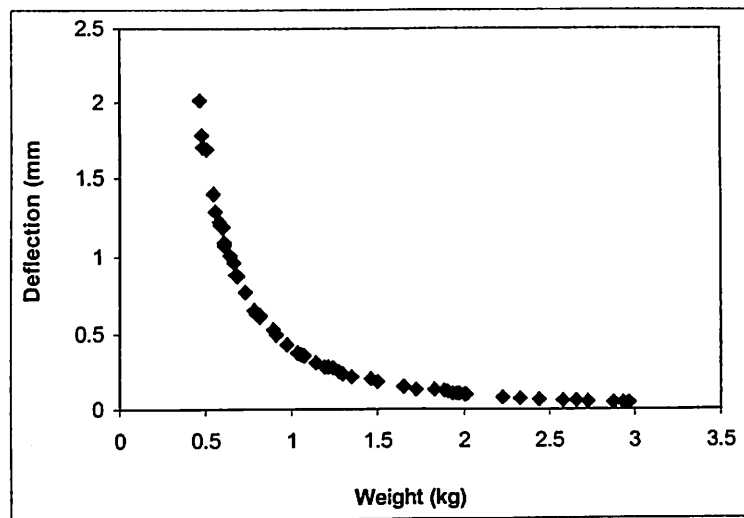
\*Literature values for maximum Deflection = 2.04, and maximum Weight = 3.06

It is clear from Table-7.4 that for both MNSDE and NSDE, the  $NPS$  increases with  $CR$  and also the  $NPS$  value is same. At  $CR$  value of 0.5,  $NPS$  is almost double the value at  $CR = 0.1$ . Further increase in  $CR$  dose not increases  $NPS$  significantly. At  $CR = 0.5$ , the MNSDE is closer to literature value for maximum deflection and weight rather than NSDE.

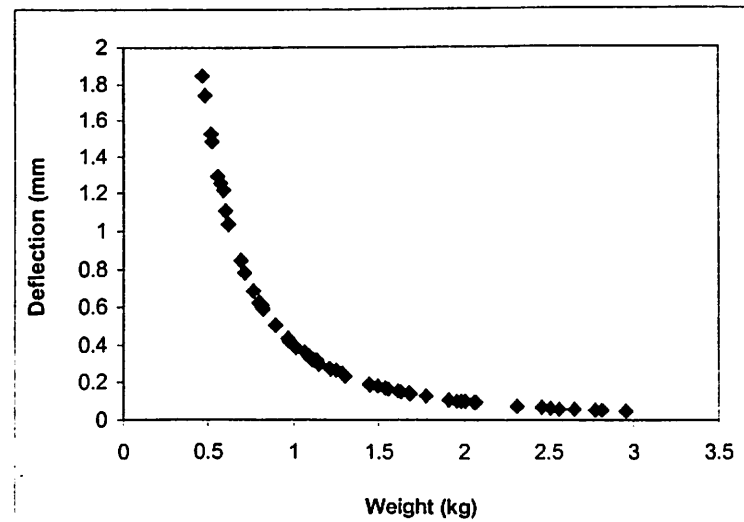




(a).  $CR = 0.1$



(b).  $CR = 0.5$



(c).  $CR = 1.0$

Fig. 7.22. Effect of  $CR$  on shape of Pareto front for cantilever design problem (MNSDE)

## 7.4.2. Conclusions

The two test problems are solved using MNSDE and results are also compared with those obtained using NSDE. The effect of various parameters ( $Max\_gen$ ,  $CR$ , and  $F$ ) is discussed and analyzed. It is observed that  $NPS$  increases with  $Max\_gen$  up to a certain value which is problem dependent. For Schaffer's function, it is about 100 while for cantilever design problem it is about 900. The effect of  $CR$  is significant in case of cantilever design problem. A high value of  $CR$  ( $\geq 0.7$ ) is found suitable. It is important to note that scaling factor not only affects the  $NPS$  but also the shape and spread of Pareto optimal front.

## 7.5. Overall Conclusions

### 7.5.1. Schaffer's function

The effect of  $Max\_gen$  is same for both the algorithms.  $CR$  does not affect the  $NPS$  as well as shape & spread of Pareto front for given value of  $F$ ,  $Max\_gen$  and seed. This is found for both the algorithms. Effect of  $F$  on  $NPS$  is more pronounced in MNSDE as compared to NSDE. However for both the algorithms, the spread of Pareto front is different for different values of  $F$  for same seed value.

### 7.5.2. Cantilever Design Problem

In both the algorithms,  $NPS$  increases with  $Max\_gen$  till a certain value. But this certain value is problem dependent. Also, in the two algorithms,  $F$  not only affects the spread but also the  $NPS$ . Lower value of  $F$  (0.1, 0.2, and 0.3) is found to give higher  $NPS$  for MNSDE while for NSDE, best value is 0.5. It is found that the parameter  $CR$  not only affects the  $NPS$  but also the shape and spread of Pareto front, for the two algorithms. Use of high  $CR$  value ( $\cong 1.0$ ) is found to be beneficial in the two algorithms.

It is important to note that even though all controlling parameters are same still the spread of Pareto front can be different in the two algorithms.

Based on the above discussion, it is recommended to use a high  $CR$  value for both NSDE & MNSDE and lower value of  $F$  for MNSDE and a value of 0.5 for NSDE.  $Max\_gen$  is found to be problem dependent.

---

# CHAPTER 8

---

## CONCLUDING REMARKS

This study demonstrates the successful application of an evolutionary computation method, i.e., Differential Evolution (DE) to benchmark test functions followed by the various types of optimization problems e.g. linear programming problem, non-linear programming problems, mixed integer non-linear programming problems (process synthesis and design), dynamic optimization problem generally encountered in chemical engineering. Also, the DE is extended to solve multi-objective optimization. In this chapter first a brief summary of the present work is presented followed by conclusions, major contributions, and future scope for research.

### **8.1. Summary**

#### **8.1.1. Introduction**

In 1980, optimization of engineering problems beyond linear programming was often viewed as a curious novelty without much benefit. Now optimization has

become a major enabling area in process systems engineering. The optimization applications are essential in all areas of process systems engineering including design, identification, control, estimation, scheduling and planning. It has evolved from a methodology of academic interest into a technology that has made, and continues to make, significant impact in the industry.

As we know, most of the traditional optimization techniques based on gradient methods have the possibility of getting trapped at local optimum depending upon the degree of non-linearity and the value of initial guess. Hence, these traditional optimization techniques do not ensure global optimum and also have limited applications. Now-a-days, non-traditional search and optimization methods (also called Evolutionary Algorithms) that often rely on physical analogies in order to generate trial points that mimic the approach to an equilibrium condition, are becoming popular. Evolutionary Algorithms (EAs) have been widely used in science and engineering for solving complex problems. An important goal of research on evolutionary algorithms is to understand the class of problems for which EAs are most suited, and, in particular, the class of problems on which they out perform other search algorithms. In the present study, DE, an evolutionary optimization technique (which is simple, fast, and robust) is chosen for application to chemical engineering problems. Further the modifications of original DE are proposed and implemented. Modifications lead to the improvement in the performance of DE.

#### **8.1.2. Comparison of DE with GA**

A comparison of GA and DE is made on the basis of type of representation used for decision variables, crossover and mutation operations, and the sequence/order of operators used. Further the application of DE to Himmelblau function showed that performance of DE is much better than that of GA. And DE has great potential and

can be applied to advantage in all the highly non-linear and complex engineering problems.

### 8.1.3. Modified Differential Evolution (MDE)

Application of DE to various test functions and selected chemical processes further proves the fact that DE is fast and robust. When applied to the problem of optimal design of ammonia synthesis reactor, the new optimum reactor length and hence objective function value (profit) is obtained. Also, the results indicate that the profiles of temperatures and flow rate are smooth and there is no reverse reaction effect as reported in Upreti and Deb (1997).

DE takes large computational time for problems involving computationally expensive objective functions. In order to reduce the computational efforts, a modified DE (MDE) and two new strategies are proposed and tested on several benchmark test functions followed by nonlinear chemical processes. Also seven test problems on process synthesis and design have been solved using DE and MDE in the present work. Performance of various algorithms (e.g. GA, M-SIMPSA, M-SIMPSA-pen, ES ( $\mu+\lambda$ ), DE, and MDE) is compared in terms of number of function evaluations (speed) and number of experiments converged to global optimum (robustness). Also, the two approaches for handling binary/discrete variables along with two methods for handling bound violations are studied and compared. It is important to note that for problems where global optimum is located on the bound of decision variables, forced bound method outperform the method without forcing the bound (e.g. Problem-7 in Chapter-4). However for problems where local optimum is located on the bound of decision variables, the method without forcing the bound is found to be robust (e.g. Problem-3 in Chapter-4). Also, the Approach-1 for handling binary variables is found to be better than that of Approach-2. This is because of

addition of constraints (binary or discrete variable is modeled as continuous variable) and addition of nonconvexities to the problem in case of Approach-2. MDE outperform all other algorithms for process synthesis and design problems.

#### **8.1.4. Hybrid Differential Evolution (HDE)**

Search space for general optimization problem can be divided into several convex regions and each region has a local optima. Classical gradient-based optimization methods show better performance than evolutionary algorithms (EAs) in finding the minimum of a convex region. However, they are not good at finding the global optima of the problem with multiple local optima. This drawback could be overcome by using EA in combination with classical gradient method. Hence a hybrid differential evolution (HDE) algorithm is proposed and tested on several benchmark test functions and further evaluated for selected nonlinear chemical engineering problems. This new method is a hybrid of DE and quasi-Newton method. A comparison of HDE with DE is made in terms of CPU-time, AAD, and success rate. It is found that the two algorithms (viz., DE and HDE) are reliable in locating the global optima of the problems studied. The HDE took less CPU-time as compared to DE. Also, HDE located the global optimum with high accuracy (very low AAD values) as compared to DE. Overall, the performance of HDE is found to be better than that of DE. Especially, for high dimension problems, the performance of HDE is significantly better than that of DE. HDE is also compared with other algorithms reported in literature such as ECTS, TS-QN, and GA.

#### **8.1.5. New Strategies of DE**

In the present work, two new strategies have been proposed and their performance is evaluated using two test functions and three nonlinear chemical engineering problems. Proposed strategies are tuned hopefully to their best control parameters.

Using these tuned control parameters, three selected nonlinear chemical engineering problems are solved. It is found that like DE and MDE, NS-1 and NS-2 also take less computational time for forced bound method as compared to method without forcing the bound. Also, none of the methods is able to locate the global optimum using method without forcing the bound for the MPBP problem. NS-1 is found to be computationally efficient for the dynamic optimization of a batch reactor and RND problem as compared to NS-2. NS-2 is found to take least CPU-time for MPBP problem while MDE is found computationally efficient for RND problem. For RND problem NS-2 is found to be robust over a wide range of key parameters as compared to other methods (DE, MDE, and NS-1). Overall, the proposed strategies (NS-1 and NS-2) are found to be competitive with DE and MDE algorithms.

#### **8.1.6. Extension of DE and MDE for MOOPs**

DE and MDE are extended for solving multi-objective optimization problems. The extended algorithms are named as NSDE and MNSDE. Two problems, one standard test problem and cantilever design problem are solved using the proposed algorithms. The results indicate that NSDE is able to locate the global Pareto front for two test problems studied. The effects of key parameters of NSDE and MNSDE, i.e., *Max\_gen*, *CR*, and *F* are discussed and analyzed. It is found that for both the test problems an increase in *Max\_gen* increases the *NPS* up to a certain value, and after that there is no significance increase in *NPS*. This certain value is problem dependent and is found to be different for the two problems studied. For Schaffer's function it is about 200, while for cantilever design problem it is about 500 in case of NSDE while it is 100 and 900 respectively for Schaffer's function and cantilever design problem using MNSDE. The effect of *CR* is significant in case of cantilever design problem while it does not affect the Schaffer's function. A high value of *CR* ( $\cong 1$ ) is found



Using these tuned control parameters, three selected nonlinear chemical engineering problems are solved. It is found that like DE and MDE, NS-1 and NS-2 also take less computational time for forced bound method as compared to method without forcing the bound. Also, none of the methods is able to locate the global optimum using method without forcing the bound for the MPBP problem. NS-1 is found to be computationally efficient for the dynamic optimization of a batch reactor and RND problem as compared to NS-2. NS-2 is found to take least CPU-time for MPBP problem while MDE is found computationally efficient for RND problem. For RND problem NS-2 is found to be robust over a wide range of key parameters as compared to other methods (DE, MDE, and NS-1). Overall, the proposed strategies (NS-1 and NS-2) are found to be competitive with DE and MDE algorithms.

#### **8.1.6. Extension of DE and MDE for MOOPs**

DE and MDE are extended for solving multi-objective optimization problems. The extended algorithms are named as NSDE and MNSDE. Two problems, one standard test problem and cantilever design problem are solved using the proposed algorithms. The results indicate that NSDE is able to locate the global Pareto front for two test problems studied. The effects of key parameters of NSDE and MNSDE, i.e., *Max\_gen*, *CR*, and *F* are discussed and analyzed. It is found that for both the test problems an increase in *Max\_gen* increases the *NPS* up to a certain value, and after that there is no significance increase in *NPS*. This certain value is problem dependent and is found to be different for the two problems studied. For Schaffer's function it is about 200, while for cantilever design problem it is about 500 in case of NSDE while it is 100 and 900 respectively for Schaffer's function and cantilever design problem using MNSDE. The effect of *CR* is significant in case of cantilever design problem while it does not affect the Schaffer's function. A high value of *CR* ( $\cong 1$ ) is found

suitable for both the problems. It is important to note that  $F$  not only affects the  $NPS$  but also the distribution of solutions in Pareto front for both the problems. It is recommended to use a high  $CR$  value for both NSDE & MNSDE and lower value of  $F$  for MNSDE and a value of 0.5 for NSDE.  $Max\_gen$  is found to be problem dependent.

## 8.2. Conclusions

Based on the above discussion the following conclusions are drawn:

1. Performance of DE is found to be better than simple GA.
2. DE is successfully applied to various benchmark test functions and nonlinear chemical engineering problems.
3. In case of optimal design of ammonia synthesis reactor, new value of optimum reactor length and profit function is found. Also, a possible error in the NAG routine is found.
4. A Modification in original DE (MDE) is proposed and tested on several test functions and nonlinear chemical processes. It is found that the performance of MDE is better than that of DE.
5. Further the MDE is tested on several test problems on process synthesis and design. The performance of MDE is compared with various other evolutionary methods such as Genetic algorithm, Evolution strategies, MINLP Simplex simulated annealing (M-SIMPASA) with or without penalty. It is found that performance of MDE is better than that of other methods.
6. A hybrid differential evolution (HDE) is proposed and tested on several benchmark test functions followed by nonlinear chemical processes. HDE is

found to take less computational time. Also, it is able to locate the global optimum with a high accuracy better than DE.

7. Also, two new strategies (NS-1 and NS-2) are proposed and tested on several benchmark test functions and difficult non-linear chemical engineering problems. NS-2 is found to be robust over the large range of key parameters. NS-1 shows a significant saving in computational time for problems having computationally expensive objective function.
8. Further a simple extension of DE and MDE algorithms for solving multi-objective optimization problems is proposed and tested on two benchmark test functions. Key parameters are tuned for these two test problems.
9. The proposed new evolutionary algorithms (MDE and HDE), new strategies of DE (NS-1 and NS-2), and new evolutionary multi-objective optimization algorithms (NSDE and MNSDE) are very useful in solving highly complex real world single and multi-objective optimization problems. And the performance of all these newly developed algorithms in this study is found to outperform the existing evolutionary algorithms.

### **8.3. Major Contributions**

1. Application of DE to various types of chemical engineering problems, i.e., linear programming problem, nonlinear programming problems, mixed integer nonlinear programming problems, dynamic optimization problem etc.
2. A new approach for handling binary variable is proposed and tested.
3. New results are obtained for Ammonia synthesis reactor problem. A possible error in the old NAG routine is identified.

4. A modified differential evolution (MDE) is proposed and evaluated for various types of problems encountered in chemical engineering.
5. A better solution (possibly global optimum) is obtained for isothermal CSTR design, Alkylation process optimization, and Fuel allocation in power plant problems.
6. A hybrid DE is proposed and evaluated.
7. Two new strategies are proposed and evaluated their performances on difficult nonlinear problems.
8. An extension of DE and MDE for solving the multi-objective optimization problems and tuning of key parameters of NSDE and MNSDE using two test problems.

#### **8.4. Future Scope for Research**

1. The idea of MDE can be extended to other DE strategies.
2. Application of DE and MDE to complex problems involving CFD simulations.
3. Application of HDE and two new strategies to other branches of engineering and comparison of MDE and HDE.
4. An improvement in MDE, HDE, and two new strategies to further meet the demand of reducing the computational efforts particularly for the problems involving time consuming simulation for evaluation of objective function.
5. Extensive analysis, the comparison of performance with other techniques, and application of NSDE and MNSDE not only to chemical engineering problems but also to other branches of engineering.

4. A modified differential evolution (MDE) is proposed and evaluated for various types of problems encountered in chemical engineering.
5. A better solution (possibly global optimum) is obtained for isothermal CSTR design, Alkylation process optimization, and Fuel allocation in power plant problems.
6. A hybrid DE is proposed and evaluated.
7. Two new strategies are proposed and evaluated their performances on difficult nonlinear problems.
8. An extension of DE and MDE for solving the multi-objective optimization problems and tuning of key parameters of NSDE and MNSDE using two test problems.

#### **8.4. Future Scope for Research**

1. The idea of MDE can be extended to other DE strategies.
2. Application of DE and MDE to complex problems involving CFD simulations.
3. Application of HDE and two new strategies to other branches of engineering and comparison of MDE and HDE.
4. An improvement in MDE, HDE, and two new strategies to further meet the demand of reducing the computational efforts particularly for the problems involving time consuming simulation for evaluation of objective function.
5. Extensive analysis, the comparison of performance with other techniques, and application of NSDE and MNSDE not only to chemical engineering problems but also to other branches of engineering.

---

## REFERENCES

---

- Abbass, H. A., Sarkar, R., and Newton, C. (2001). PDE: A Pareto-frontier differential evolution approach for multi-objective optimization problems. *Proceedings of IEEE congress on Evolutionary Computation*, 971-978.
- Adjiman, C. S., Dallwig, S., Floudas, C. A., and Neumaier, A. (1998a). A global optimization method,  $\alpha$ BB, for general twice-differentiable constrained NLPs – I. Theoretical advances. *Computers & Chemical Engineering*, 22 (9), 1137-1158.
- Adjiman, C. S., Androulakis, I. P., and Floudas, C. A. (1998b). A global optimization method,  $\alpha$ BB, for general twice-differentiable constrained NLPs – II. Implementation and computational results. *Computers & Chemical Engineering*, 22 (9), 1159-1179.
- Adjiman, C. S., Androulakis, I. P., and Floudas, C. A. (2000). Global optimization of mixed integer nonlinear problems. *AIChE Journal*, 46 (9), 1769-1797.
- Adjiman, C. S., Androulakis, I. P., and Floudas, C. A. (1997). Global optimization of MINLP problems in process synthesis and design. *Computers & Chemical Engineering*, 21 (Suppl.), S445-S450.
- Al-Khayyal, F. A. (1992). Generalized bilinear programming. Part I. Models, applications and linear programming relaxation. *European Journal of Operations Research*, 60, 306 – 314.

- Al-Khayyal, F. A. and Falk, J. E. (1983). Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8, 273 – 286.
- Androulakis, I. P. and Venkatasubramanian, V. (1991). A genetic algorithm framework for process design and optimization. *Computers & Chemical Engineering*, 15(4), 217-228.
- Angira, R. and Babu, B. V. (2005a). Non-dominated Sorting Differential Evolution (NSDE): An Extension of Differential Evolution for Multi-objective Optimization. Accepted and to be presented at the 2<sup>nd</sup> Indian International Conference on Artificial Intelligence (IICAI-2005), December 20-22, Pune, India.
- Angira, R. and Babu, B. V. (2005b). Optimization of Non-linear chemical processes using Modified Differential Evolution (MDE). Accepted and to be presented at the 2<sup>nd</sup> Indian International Conference on Artificial Intelligence (IICAI-05), Pune, India, December 20-22.
- Angira, R. and Babu, B. V. (2005c). Process Synthesis and Design using Modified Differential Evolution (MDE). To be presented at *International Symposium & 58th Annual Session of IChE (CHEMCON-2005)*, December 14 – 17, New Delhi.
- Angira, R. and Babu, B. V. (2005d). Simulation and Optimal Design of Ammonia Synthesis Reactor. Communicated. *Chemical Engineering Research and Design*.
- Angira, R. and Babu, B. V. (2003). Evolutionary computation for global optimization of non-linear chemical engineering processes. *Proceedings of International symposium on process systems engineering & control (ISPSEC '03)-for productivity enhancement through design and optimization*, 87-91, IIT-Bombay, Mumbai, India.
- Annable, D. (1952). Application of the Temkin Kinetic Equation to Ammonia Synthesis in Large – Scale Reactors. *Chemical Engineering Science*, 1(4), 145.
- Babu, B. V. (2004). *Process Plant Simulation*. Oxford University Press, India.
- Babu, B. V. and Anbarasu, B. (2005). Multi-Objective Differential Evolution (MODE): An Evolutionary Algorithm for Multi-Objective Optimization Problems (MOOPs). To be presented at The Third International Conference on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS-2005), Singapore, December 13-16, 2005.

- Babu, B. V. and Angira, R. (2001a). Optimization of non-linear functions using evolutionary computation. *Proceedings of 12th ISME conference*, pp. 153-157, Jan, 10-12, Chennai, India.
- Babu, B. V. and Angira, R. (2001b). Optimization of thermal cracker operation using differential evolution. *Proceedings of International symposium & 54th annual Session of IChE (CHEMCON-2001)*, December 19-22, Chennai, India.
- Babu, B. V. and Angira, R. (2002a). Optimization of Non-Linear Chemical Processes Using Evolutionary Algorithm. *Proceedings of International Symposium & 55<sup>th</sup> Annual session of IChE (CHEMCON-2002)*, December 19-22, OU, Hyderabad, India.
- Babu, B. V. and Angira, R. (2002b). A Differential Evolution Approach for Global Optimization of MINLP Problems. *Proceedings of 4<sup>th</sup> Asia Pacific Conference on Simulated Evolution and Learning (SEAL-2002)*, Singapore, November 18-22, Vol. 2, pp. 880-884.
- Babu, B. V. and Angira, R. (2003a). Optimization of Water Pumping System Using Differential Evolution Strategies. *Proceedings of The Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS-2003)*, Singapore, December 15-18, 2003.
- Babu, B. V. and Angira, R. (2003b). New Strategies of Differential Evolution for Optimization of Extraction Process. *Proceedings of International symposium & 56th annual session of IChE (CHEMCON-2003)*, December 19-22, Bhubhaneswar, India.
- Babu, B. V. and Angira, R. (2004). Optimization Using Hybrid Differential Evolution Algorithms. *Proceedings of International Symposium & 57th Annual Session of IChE (CHEMCON-2004)*, December 27-30, Mumbai.
- Babu, B. V. and Angira, R. (2005a). Optimal Design of an Auto-thermal Ammonia Synthesis Reactor. *Computers & Chemical Engineering*, 29 (5), 1041-1045.
- Babu, B. V. and Angira, R. (2005b). Modified Differential Evolution (MDE) for Optimization of Non-Linear Chemical Processes Using", Communicated. *Computers & Chemical Engineering*.
- Babu, B. V. and Chaturvedi, G. (2000). Evolutionary computation strategy for optimization of an alkylation reaction. *Proceedings of International symposium & 53rd Annual Session of IChE (CHEMCON-2000)*, Calcutta, India.



- Babu, B. V. and Chaurasia, A. S. (2003a). Optimization of pyrolysis of biomass using differential evolution approach. *Proceedings of Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS-2003)*, Singapore.
- Babu, B. V. and Chaurasia, A. S. (2003b). Modeling, Simulation, and Estimation of Optimum Parameters in Pyrolysis of Biomass. *Energy Conversion and Management*, 44 (13), 2135-2158.
- Babu, B. V. and Chaurasia, A. S. (2004a). Parametric Study of Thermal and Thermodynamic Properties on Pyrolysis of Biomass in Thermally Thick Regime. *Energy Conversion and Management*, 45 (1), 53-72.
- Babu, B. V. and Chaurasia, A. S. (2004b). Dominant Design Variables in Pyrolysis of Biomass Particles of Different Geometries in Thermally Thick Regime. *Chemical Engineering Science*, 59 (3), 611-622.
- Babu, B. V. and Chaurasia, A. S. (2004c). Heat Transfer and Kinetics in the Pyrolysis of Shrinking Biomass Particle. *Chemical Engineering Science*, 59 (10), 1999-2012.
- Babu, B. V. and Gautam, K. (2001). Evolutionary computation for scenario-integrated optimization of dynamic systems. *Proceedings of International symposium & 54th annual session of IChE (CHEMCON-2001)*, Chennai, India.
- Babu, B. V. and Jehan, M. M. L. (2003). Differential Evolution for Multi-Objective Optimization. *Proceedings of International Conference on Evolutionary Computation (CEC-2003)*, Canberra, Australia, December 8-12, 2003, 2696-2703.
- Babu, B. V. and Mohiddin, S. B. (1999). Automated Design of Heat Exchangers Using Artificial Intelligence based optimization. *Proceedings of 52<sup>nd</sup> Annual session of IChE (CHEMCON-99)*, Chandigarh, India, December 20 – 23, 1999.
- Babu, B. V. and Munawar, S. A. (2000). Differential Evolution for the Optimal Design of Heat Exchangers. *Proceedings of All-India seminar on Chemical Engineering Progress on Resource Development: A Vision 2010 and Beyond*, IE (I), Bhubaneswar, India, March 11, 2000.
- Babu, B. V. and Munawar, S. A. (2001). Optimal design of shell & tube heat exchanger by different strategies of differential evolution. *PreJournal.com–The Faculty Lounge, Article No. 003873, posted on website Journal <http://www.prejournal.com>*.

- Babu, B. V. and Sastry, K. K. N. (1999). Estimation of Heat-Transfer Parameters in a Trickle-Bed Reactor using Differential Evolution and Orthogonal Collocation. *Computers & Chemical Engineering*, 23, 327 – 339.
- Babu, B. V. and Singh, R. P. (2000). Synthesis & optimization of heat integrated distillation systems using differential evolution. *Proceedings of all-India seminar on chemical engineering progress on resource development: a vision 2010 & beyond*, IE (I), Bhubaneswar, India.
- Babu, B. V. and Vivek, N. (1999). Genetic algorithms for estimating heat transfer parameters in trickle bed reactors. *Proceedings of 52<sup>nd</sup> Annual session of IChE (CHEMCON-99)*, Chandigarh, India, December 20 – 23, 1999.
- Babu, B. V., Chakole, P. G., and Mubeen, J. H. S. (2005a). Multiobjective Differential Evolution (MODE) for Optimization of Adiabatic Styrene Reactor. *Chemical Engineering Science*, 60 (17), 4824-4839.
- Babu, B. V., Mubeen, J. H. S., and Chakole, P. G. (2005b). Multiobjective Optimization Using Differential Evolution. *TechGenesis-The Journal of Information Technology*, 2 (2), 4-12.
- Bäck, T. (1996), *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Inc., Oxford, 1996.
- Bergey, P. K. and Ragsdale, C. (2005). Modified differential evolution: a greedy random strategy for genetic recombination. *Omega*, 33, 255 – 265.
- Biegler, L. T. and Grossmann, I. E. (2004). Retrospective on optimization. *Computers & Chemical Engineering*, 28, 1169-1192.
- Bilbro, G. L. and Snyder, W. E. (1991). Optimization of function with many minima. *IEEE Transaction on System, Man, and Cybernetics*, 21 (4), 840 – 849.
- Box, G. E. P. (1957). Evolutionary Operation: a method of increasing industrial productivity. *Applied Statistics*, 6, 81 – 101.
- Bracken, J. and McCormick, G. P. (1968). *Selected Applications of Nonlinear Programming*. New York, John Wiley & Sons, Inc.
- Cardoso, M. F., Salcedo, R. L., Feyer de Azevedo, S., and Barbosa, D. (1997). A simulated annealing approach to the solution of MINLP problems. *Computers & Chemical Engineering*, 21, 1349-1364.
- Chakraborti, N., Misra, K., Bhatt, B., Barman, N. and Prasad, R. (2001). Tight-Binding Calculations of Si-H Clusters Using Genetic Algorithms and Related

- Techniques: Studies Using Differential Evolution. *Journal of Phase Equilibria* 22(5), 525 – 530.
- Chelouah, R. and Siarry, P. (2000). Tabu search applied to global optimization. *European Journal of Operational Research*, 123, 256 – 270.
- Chiou, J. P. and Wang, F.S. (1999). Hybrid method of evolutionary algorithms for static and dynamic optimization problems with application to a fed-batch fermentation process. *Computers & Chemical Engineering*, 23, 1277-1291.
- Chung, S. F. (1972). Mathematical model and optimization of drying process for a through-circulation dryer. *Canadian Journal of Chemical Engineering*, 50, 657 – 662.
- Chung, S. F. (1973). Letter to the Editor. *Canadian Journal of Chemical Engineering*, 51, 262.
- Ciric, A. R. and Gu, D. (1994). Synthesis of Nonequilibrium Reactive distillation processes by MINLP optimization. *American Institute of Chemical Engineers Journal*, 40, 1479 – 1487.
- Coello, C. A. (1999). A Comprehensive Survey of Evolutionary- Based Multi objective Optimization Techniques. *International Journal of Knowledge and Information Systems*, 1 (3), 269-308.
- Cordoso, J. C., Davin, A., Floquet, P., Pibouleau, L., & Domenech, S. (1997). Synthesis of optimal reactor networks using mathematical programming and simulated annealing. *Computers & Chemical Engineering*, 21, S47-S52.
- Come, D., Dorigo, M., and Glover, F. (1999). *New Ideas in optimization*. McGraw-Hill, Berkshire, England (U. K.).
- Costa, L. & Oliviera, P. (2001). Evolutionary algorithms approach to the solution of mixed integer no-linear programming problems. *Computers & Chemical Engineering*, 25, 257-266.
- Cvijovic', D. and Klinowski, J. (1995). Taboo search: an approach to the multiple minima problem. *Science*, 267, 664 - 666.
- Dadebo, S. A. and K. B. Mcauley (1995). Dynamic Optimization of Constrained Chemical Engineering problems Using Dynamic Programming. *Computers & Chemical Engineering*, 19, 513-525.
- Dasgupta, D. and Michalewicz, Z. (1997). *Evolutionary algorithms in Engineering Applications*, 3 - 23, Springer-Verlag, Berlin, Germany.

- Davis, L. (1991). *Handbook of genetic algorithms*. New York, Van Nostrand Reinhold.
- De Jong, K. A. (1992). Are genetic algorithms function optimizers? *Proceedings of the Second International conference on Parallel Problem Solving from Nature*.
- Deb, K. (1996). *Optimization for engineering design: Algorithms and examples*. New Delhi, Prentice-Hall.
- Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons.
- Deb, K., Pratap, A., Agarwal, S., Meyarivan, T. (2002). A fast and elitist multi objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6 (2), 182-197.
- Dembo, R. S. (1976). A set of geometric programming test problems and their solutions. *Math. Programming*, 10, 193-213.
- Diwekar, U. M. and Rubin, E. S. (1993). Efficient handling of the implicit constraints problem for the ASPEN MINLP Synthesizer. *Industrial & Engineering Chemistry Research*, 32, 2006-2011.
- Diwekar, U. M., Grossmann, I. E., and Rubin, E. S. (1992). An MINLP Process Synthesizer for a Sequential Modular Simulator. *Industrial & Engineering Chemistry Research*, 31, 313-322.
- Dollena S. H., Allen, D. M., and Stromberg, A. J. (2001). Determining the number of components in mixtures of linear models. *Computational Statistics & Data Analysis*, 38, 15 – 48.
- Dorigo, M., Maniezzo, A., and Colomi, A. (1996). The Ant system: Optimization by a colony of cooperating agents. *IEEE transaction on System, Man, and Cybernetics: Part B*, 26 (1), 29 – 41.
- Douglas, J. M. (1985). A hierarchical decision procedure for process synthesis. *AICHE Journal*, 31, 353–362.
- Dyson, D.C. (1965). *Optimal Design of Reactors for Single Exothermic Reversible Reactions*, Ph.D. Thesis, London University.
- Edgar, T. F. and Himmelblau, D.M. (1989). *Optimization of Chemical Processes*, McGraw-Hill, Inc., Singapore, pp. 534 – 539.
- Edgar, T. F. and Himmelblau, D.M. (2001). *Optimization of Chemical Processes*, McGraw-Hill, Inc., Singapore.

- Eymery, J. (1964). *Dynamic Behavior of an Ammonia Synthesis Reactor*. D. Sc. Thesis, M.I.T.
- Falk, J. E. and Soland, R. M. (1969). An algorithm for separable nonconvex programming problems. *Management Science*, 15, 550 – 569.
- Fan, H. Y. and Lampinen, J. (2003). A Trigonometric Mutation Operation to Differential Evolution. *Journal of Global Optimization*, 27, 105–129.
- Floudas, C. A. and Pardalos, P.M. (1990). *A collection of test problems for constrained Global optimization algorithms*. Lecture Notes in Computer Science, vol. 455. Springer, Berlin, Germany.
- Floudas, C. A. and Visweswaran, V. (1990). A global optimization algorithm (GOP) for certain classes of nonconvex NLPs – I theory. *Computers & Chemical Engineering*, 14, 1397 – 1417.
- Floudas, C. A., Aggarwal, A., and Ciric, A. R. (1989). Global optimum search for nonconvex NLP and MINLP problems. *Computers & Chemical Engineering*, 13, 1117 – 1132.
- Floudas, C. A., Ciric, A. R., and Grossmann, I. E. (1986). Automatic synthesis of optimum heat exchanger network configurations. *American Institute of Chemical Engineers Journal*, 32, 276-290.
- Floudas, C.A. (1995). *Nonlinear and mixed-integer optimization*. Oxford University Press, New York.
- Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ.
- Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, New York.
- Fonseca C. M. and Fleming, P. J. (1993). Genetic algorithms for multi objective optimization: formulation, discussion, and generalization. *Genetic Algorithms: Proceeding of the Fifth International Conference*, Forrest S. ed., 416-423.
- Fraga, E.S. and Matias, T. R. S. (1996). Synthesis and optimization of a non-ideal distillation system using a parallel genetic algorithm. *Computers & Chemical Engineering*, 20, pp. S79-S84.
- Fraser, A. S. (1957). Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science*, 10, 484 – 491.
- Glover, F. (1989). Tabu Search Part I. *ORSA Journal on Computing*, 1, 190 – 206.

- Glover, F. (1997). A template for Scatter Search and Path Relinking. In: Hao, J. K., Lutton, E., Ronald, E., Schoenauer, M. and Snyers, D. (eds.), *Lecture Notes in Computer Science*, 1363, 1 – 53.
- Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*, Reading, MA: Addison-Wesley.
- Goulcher, R. and Long, J. J. C. (1978). The solution of steady-state chemical engineering optimization problems using a random search algorithm. *Computers & Chemical Engineering*, 2(1), 33 – 36.
- Grossmann, I. E. (1985). Mixed-Integer Programming approach for the synthesis of integrated process flowsheets. *Computers & Chemical Engineering*, 9, 463.
- Grossmann, I. E. and Biegler, L. T. (2004). Part II. Future perspective on optimization. *Computers & Chemical Engineering*, 28, 1193-1218.
- Grossmann, I. E. and Sargent, R. W. H. (1979). Optimum design of multipurpose chemical plants. *Industrial & Engineering Chemistry Process Design Development*, 18, 343.
- Gupta, O. P. (1994). *Elements of Fuels, Furnaces and Refractories*. Khanna Publishers, New Delhi, 161-168.
- Hajela, P., and Lin, C. Y. (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization*, 4, 99-107.
- Hansen, E. R. (1980). Global optimization using interval analysis: The multi-dimensional case. *Numerische Mathematik*, 34, 247 – 270.
- Hansen, P., Jaumard, B., and Lu, S. (1992). Global optimization of univariate Lipschitz functions: New algorithms and computational comparison. *Mathematical Programming*, 60, 161 – 220.
- Hartland, S. and Mecklenburgh, J. C. (1975). *The Theory of Backmixing*. Wiley, New York.
- Himmelblau, D.M. (1997). *Basic Principles and Calculations in Chemical Engineering*. 6<sup>th</sup> Ed, PHI.
- Hobson, G. D. (1975). *Modern Petroleum Technology*. 4<sup>th</sup> Ed., Applied Science Publisher, Essex, Great Britain.
- Holland, J. H. (1975). *Adaptations in natural and artificial systems*. Ann Arbor: University of Michigan Press.

- Horn, J., Nafpliotis, N., and Goldberg, D.E. (1994). A niched Pareto genetic algorithm for multi objective optimization. *Proceedings of the First IEEE Conference on Evolutionary Computation*, 82 – 87.
- Horst, R., Thoi, N. V., and De Vries, J. (1992). A new simplicial cover technique in constrained global optimization. *Journal of Global Optimization*, 2, 1 – 19.
- Hovanesian, S. A. and Stout, T. M. (1963). Optimum fuel allocation in power plants. *IEEE Transaction of Power Apparatus System*, 82, 329.
- Hovanesian, S. A. and Pipes, L. A. (1969). *Digital computer methods in engineering*. McGraw-Hill, New York.
- Ilonen, J., Kamarainen, J. K., Lampinen, J. (2003). Differential Evolution Training Algorithm for Feed-Forward Neural Networks. *Neural Processing Letters* 7, 1, 93 – 105.
- Jackson, P. J. (1977). Ph.D. Thesis, Monish University, Victoria, Australia.
- Jackson, P. J. and Agnew, J. B. (1980). A Model-Based scheme for the on-line optimization of a liquid extraction process. *Computers & Chemical Engineering*, 4, 241.
- Joshi, R. & Sanderson, A. C. (1999). Minimal representation multi-sensor fusion using differential evolution. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 29 (1), 63-76.
- Kirkpatrick, S., Gelatt, C. D. & Vechhi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220 (4568), 671-680.
- Knowles, J., and Corne, D. (2000). Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary computation*, 8, 142-172.
- Kocis, G. R. and Grossmann, I. E. (1987). Relaxation strategy for the structural optimization of process flow sheets. *Industrial & Engineering Chemistry Research*, 26, 1869-1880.
- Kocis, G. R. and Grossmann, I. E. (1988). Global optimization of nonconvex mixed-integer nonlinear programming (MINLP) problems in process synthesis. *Industrial & Engineering Chemistry Research*, 27, 1407-1421.
- Kocis, G. R. and Grossmann, I. E. (1989). A modeling and decomposition strategy for the MINLP optimization of process flowsheets. *Computers & Chemical Engineering*, 13, 797-819.
- Kreyszig, E. (1993). *Advanced Engineering Mathematics*. 7<sup>th</sup> Ed, John Wiley & Sons, Inc., Singapore.

- Kursawe, F. (1991). A variant of evolution strategies for vector optimization. In Schwefel H.P. and Manner, R. eds., *Parallel problem solving from nature, First workshop proceedings*, Vol. 496 of Lecture notes in computer science, 193-197.
- Kyprianou, A., Worden, K. and Panet, M. (2001). Identification of hysteretic systems using the differential evolution algorithm. *Journal of Sound and Vibration*, 248 (2), 289 – 314.
- Lampinen, J. (2001). Solving Problems Subject to Multiple Nonlinear Constraints by the Differential Evolution. In: Radek Matoušek and Pavel Ošmera (eds.). *Proceedings of MENDEL 2001, 7th International Conference on Soft Computing*, Brno, Czech Republic. Brno University of Technology, Faculty of Mechanical Engineering, Institute of Automation and Computer Science, Brno (Czech Republic), pp. 50-57.
- Lee, M. H., Han, C., and Chang, K. S. (1999). Dynamic optimization of a continuous polymer reactor using a modified differential evolution, *Industrial & Engineering Chemistry Research*, 38, 4825-4831.
- Leibman, J., Lasdon, L., Schrage, L., and Waren, A. (1986). Modeling and optimization with GINO. *The Scientific Press*, Palo Alto, CA.
- Li, H. L. (1992). An approximate method for local optima for nonlinear mixed integer programming problems. *Computers & Operation Research*, 19 (5), 435 – 444.
- Lin, Y. C., Hwang, K. S., and Wang, F. S. (2001). Co-Evolutionary Hybrid Differential Evolution for Mixed-Integer Optimization Problems. *Engineering Optimization*, 33 (6), 663 – 682.
- Linnhoff, B. (1981). In *Foundations of Computer Aided Chemical Process Design*: Mah, R. S. H., Seider, W. D.; Eds.; Engineering Foundation, New York, Vol. II, 537 – 572.
- Logsdon, J. S. and Biegler, L. T. (1989). Accurate solution of differential algebraic equations. *Industrial & Engineering Chemistry Research*, 28, 1628-1639.
- Lopez, C. I. L., Van, W. L. G., and Van, S. G. (2001). Parameter Control Strategy in Differential Evolution Algorithm for Optimal Control. In: M.H. Hamza (ed.) (2001). *Proceedings of the IASTED International Conference Artificial Intelligence and Soft Computing (ASC 2001)*, May 21-24, 2001, Cancun , Mexico, pp. 211-216. ACTA Press, Calgary (Canada).



- Lu, J.C. and Wang, F.S. (2001). Optimization of Low Pressure Chemical Vapor Deposition Reactors Using Hybrid Differential Evolution. *Canadian Journal of Chemical Engineering*, 79 (2), 246-254.
- Luss, R. and Jaakola, T. H. I. (1973). Optimization by direct search and systematic reduction of the size of search region. *American Institute of Chemical Engineers Journal*, 19, 760 – 766.
- Manish, C.T., Yan Fu, and Urmila, M. D. (1999). Optimal design of heat exchangers: A genetic algorithm framework. *Industrial & Engineering Chemistry Research*, 38, 456- 467.
- Maranas, C. D. and Floudas, C. A. (1997). Global optimization in generalized geometric programming. *Computers & Chemical Engineering*, 21 (4), 351-370.
- McCabe, W. L., Smith, J. C., and Harriott, P. (1993). *Unit Operations of Chemical Engineering*. 5<sup>th</sup> ed. McGraw-Hill, New York.
- McLeod, A.S., Johnston, M. E., and Gladden, L. F. (1997). Development of a genetic algorithm for molecular scale catalyst design. *Journal of Catalysis*, 167, 279-285.
- Muhlenbein, H., Schomisch, M. and Born, J. (1991). The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17, 619–632.
- Munawar, S. A. and Gudi, R. (2004). A Nonlinear transformation based MINLP Solution Method. *Proceedings of International Symposium & 57th Annual Session of IChE (CHEMCON-2004)*, December 27-30, Mumbai.
- Murase, A., Roberts, H.L., and Converse, A.O. (1970). Optimal Thermal Design of an Auto thermal Ammonia Synthesis Reactor. *Industrial & Engineering Chemistry Process Design Development*, 9 (4), 503- 513.
- NAG, *Web site of Numerical Algorithm Group* as on July, 2004. <http://www.nag.com/numeric/FL/manual/html/genint/FLwithdrawn.asp>
- Nishida, N., Stephanopoulos, G., and Westerberg, A. W. (1981). Journal Review: Process synthesis. *American Institute of Chemical Engineers Journal*, 27, 321.
- Norman, M. G. and Moscato, P. (1991). A competitive and cooperative Approach to complex combinatorial search. *Proceedings of the 20<sup>th</sup> Informatics and Operations Research Meeting, Buenos Aires (20<sup>th</sup> JAIIO)*, pp. 3.15 – 3.29.
- Onwubolu, G. C. and Babu, B. V. (2004). *New Optimization Techniques in Engineering*. Springer-Verlag, Heidelberg, Germany.
- Papoulias, S. and Grossmann, I. E. (1983). A structural optimization approach in process synthesis, Parts I – III. *Computers & Chemical Engineering*, 7, 695 – 734.

- Price, K., and Storn, R. (1997). Differential Evolution-a simple evolution strategy for fast optimization. *Dr. Dobb's Journal*, 22, 18-24 and 78.
- Price, K., and Storn, R. (2005). *Home page of differential evolution as on August 25*. URL: <http://www.ICSI.Berkeley.edu/~storn/code.html>
- Quesada, I. and Grossmann, I. E. (1995). A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6, 39 – 76.
- Ray, T. (1991). An approach to the synthesis of life. *Artificial Life II* Ed. Langton, C. G., Taylor, C., Farmer, J. D., and Rasmussen, S. (Reading, MA: Addison-Wesley) pp. 371–408.
- Ray, W. H. (1981). *Advanced Process control*, McGraw-Hill, New York.
- Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann – Holzboog, Stuttgart.
- Renfro, J. G., Morshedi, A. M. and Osbjornsen, O.A. (1987). Simultaneous optimization and solution of systems described by differential/algebraic equations. *Computer and Chemical Engineering*, 11, 503-517.
- Rosenbrock, H.H., and Storey, C. (1966). *Computational Techniques for Chemical Engineers*. Pergamon Press, London.
- Rudd, D. F., Powers, G. J., Siirola, J. J. (1973). *Process Synthesis*. Prentice-Hall: Englewood Cliffs, NJ.
- Rudolph, G. (1992). On correlated mutations in evolutionary strategies. In Männer, R., Manderick, B. (eds.), *Parallel Problem solving from Nature II*, Elsevier Science Press, 105 – 114.
- Rudolph, G. (1996). Convergence of evolutionary algorithms in general search space. *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, IEEE Press: New York; 50–54.
- Rutenbar, R. A. (1989). Simulated annealing algorithms: An overview. *IEEE Circuits & Devices Magazine*, 100, 19-26.
- Ryoo, H. S. and Sahinidis, N. V. (1995). Global optimization of nonconvex NLPs and MINLPs with Applications in Process Design. *Computers & Chemical Engineering*, 19(5), 551-566.
- Salcedo, R.L. (1992). Solving Nonconvex Nonlinear Programming Problems with Adaptive Random Search. *Industrial & Engineering Chemistry Research*, 31, 262.

- Sastry, K. K. N., Behra, L., and Nagrath, I. J. (1998). Differential evolution based fuzzy logic controller for nonlinear process control. *Fundamenta Informaticae: Special Issue on Soft Computation*.
- Sauer, R. N., Coville, A. R., and Burwick, C. W. (1964). Computer Points Way to More Profits. *Hydrocarbon Processing Petroleum Refiner*, 43, 84.
- Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithm. *Proceeding of the First International Conference on Genetic Algorithms*, 93-100.
- Schwefel, H. P. (1981). *Numerical Optimization of computer models*. New York: John Wiley & Sons.
- Schwefel, H. P. (1995). *Evolution and Optimum seeking*. New York: John Wiley & Sons.
- Shah, M. J. (1967). Control Simulation in Ammonia Production. *Industrial & Engineering Chemistry*, 59, 72.
- Sherali, H. D. and Alameddine, A. (1992). A new reformulation – linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2, 379 – 410.
- Sherali, H. D. and Tuncbilek, C. H. (1992). A global optimization algorithm for polynomial programming problems using a reformulation – linearization technique. *Journal of Global Optimization*, 2, 101 – 112.
- Siarry, P. and Berthiau, G. (1997). Fitting of tabu search to optimize functions of continuous variables. *International Journal for Numerical Methods in Engineering*, 40, 2449 - 2457.
- Singh, C. P. P. and Saraf, D. N. (1979). Simulation of Ammonia Synthesis Reactors. *Industrial & Engineering Chemistry Process Design Development*, 18 (3), 364 – 370.
- Smith, E. M. B. and Pantelides, C. C. (1999). A symbolic reformulation/spatial branch and bound algorithm for the global optimization of nonconvex MINLPs. *Computers & Chemical Engineering*, 23, 457 – 478.
- Soland, R. M. (1971). An algorithm for separable nonconvex programming problems. II. Nonconvex constraints. *Management Science*, 17, 759 – 773.
- Sourander, M. L., Kolari, M., Cugini, J. C., Poje, J. B., and White, D. C. (1984). Control and Optimization of Olefin-Cracking Heaters. *Hydrocarbon Processing*. 63. (June, 1984).

- Spears, W. M., De Jong, K. A., Bäck, T., Fogel, D. B., and de Garis, H. (1993). An Overview of Evolutionary Computation. *Machine Learning: ECML-93, European Conference on Machine Learning, Lecture Notes in Artificial Intelligence*, No. 667, Ed. Brazdil, P.B., pp442-459.
- Srinivas, N. and Deb, K. (1995). Multi-objective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2 (3), 221-248.
- Stephanopoulos, G. and Westerberg, A. W. (1976). Studies in process synthesis: Part II. Evolutionary synthesis of optimal process flowsheets. *Chemical Engineering Science*, 31, 195-204.
- Stoecker, W. F. (1971). *Design of Thermal Systems*. 3<sup>rd</sup> ed., McGraw-Hill International edition, Singapore, pp 117-121.
- Storn, R. (1995). Differential Evolution design of an IIR-filter with requirements for magnitude and group delay. *International Computer Science Institute*, TR-95-026.
- Taha, H. A. (1997). *Operations Research – An Introduction*". Prentice-Hall, New Delhi, 67-89.
- Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P., and Vrahatis, M. N. (2004). Parallel Differential Evolution. *Proceedings of Congress on Evolutionary Computation (CEC - 2004)*. Available at <http://www.math.upatras.gr/~dtas/papers/TasoulisPPV2004.pdf>.
- Tuy, H., Thieu, T. V., and Thai, N. Q. (1985). A conical algorithm for globally minimizing a concave function over a closed convex set. *Mathematics of Operations Research*, 10, 498 – 514.
- Umeda, T. and Ichikawa, A. (1971). A modified complex method for optimization. *Industrial & Engineering Chemistry Process Design Development*, 10, 229-236.
- Upreti, S.R. and Deb, K. (1997). Optimal Design of an Ammonia Synthesis Reactor using Genetic Algorithms. *Computers & Chemical Engineering*, 21, 87-92.
- Venkatasubramanian, V., Chan, K., and Caruthers, J.M. (1994). Computer-aided molecular design using genetic algorithms. *Computers & Chemical Engineering*, 18, 833-844.
- Wang, F. S. & Cheng, W. M. (1999). Simultaneous optimization of feeding rate and operation parameters for fed-batch fermentation processes. *Biotechnology Progress*, 15(5), 949-952.

- Wang, F. S. and Chiou, J. P. (1997). Optimal control and optimal time location problems of differential- algebraic systems by differential evolution. *Industrial & Engineering Chemistry Research*, 36, 5348-5357.
- Wang, F. S., Jing, C. H., and Tsao, G. T. (1998). Fuzzy-decision-making problems of fuel ethanol production using genetically engineered yeast. *Industrial & Engineering Chemistry Research*, 37 (8), 3434-3443.
- Wang, F. S., Su, T. L., and Jang, H. J. (2001). Hybrid Differential Evolution for Problems of Kinetic Parameter Estimation and Dynamic Optimization of an Ethanol Fermentation Process. *Industrial and Engineering Chemistry Research*, 40 (13), 2876 – 2885.
- Xue, F., Sanderson, A. C., Graves, R. J. (2003). Pareto-based multi-objective differential evolution. *Proceedings of the Congress on Evolutionary Computation (CEC'2003)*, Vol. 2, Canberra, Australia, IEEE Press, 862 – 869.
- Zaharie, D. (2002). Critical Values for the Control Parameters of Differential Evolution Algorithms. In: Matoušek, Radek and Ošmera, Pavel (eds.). *Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing*, June 5 – 7, Brno, Czech Republic, pp. 62 – 67.
- Zamora, J. M. and Grossmann, I. E. (1999). A branch and bound algorithm for problems with concave univariate. *Journal of Global Optimization*, 14, 217 – 249.
- Zelinka, I. and Lampinen, J. (2000). SOMA – Self –Organizing Migrating Algorithm. *6<sup>th</sup> International Conference on soft computing, Mendel 2000*, Brno, Czech Republic, ISBN 80-214-1609-2.
- Zitzler, E., and Thiele, L. (1999). Multi-objective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3, 257-271.

---

## LIST OF PUBLICATIONS

---

### Research Publications in International Journals:

1. Babu, B. V. and Angira, R., "Optimal design of an auto-thermal ammonia synthesis reactor", *Computers & Chemical Engineering*, Vol. 29 (No. 5), pp. 1041-1045, 2005.
2. Babu, B. V. and Angira, R., "Modified Differential Evolution (MDE) for Optimization of Non-Linear Chemical Processes", Accepted. *Computers & Chemical Engineering*, 2005.

### Research Papers Communicated to International Journals:

3. Angira, R. and Babu, B. V., "Simulation and Optimal Design of Ammonia Synthesis Reactor", Communicated. *Chemical Engineering Research and Design*, 2005.

### Research Publications in National Conferences / Proceedings:

4. Babu, B. V. and Angira, R., "Optimization of Non-linear functions using Evolutionary Computation". *Proceedings of 12<sup>th</sup> ISME Conference*, Chennai, India, Jan 10-12, 153-157, 2001.

### Research Publications in International Conferences / Proceedings:

5. Babu, B. V. and Angira, R., "Optimization of thermal cracker operation using Differential Evolution". *Proceedings of International Symposium & 54th Annual Session of IChE (CHEMCON-2001)*, Chennai, December 19-22, 2001.
6. Babu, B. V. and Angira, R., "A Differential Evolution Approach for Global Optimization of MINLP Problems". *Proceedings of 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02)*, Singapore, November 18-22, Vol. 2, pp 866-870, 2002.
7. Babu, B. V. and Angira, R., "Optimization of Non-Linear Chemical Processes Using Evolutionary Algorithm". *Proceedings of International Symposium & 55th Annual Session of IChE (CHEMCON-2002)*, OU, Hyderabad, December 19-22, 2002.

8. Babu, B. V. and Angira, R., "Optimization of Water Pumping System Using Differential Evolution Strategies". *Proceedings of The Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems (CIRAS-2003)*, Singapore, December 15-18, 2003.
9. Babu, B. V. and Angira, R., "New Strategies of Differential Evolution for Optimization of Extraction Process". *Proceedings of International Symposium & 56th Annual Session of IChE (CHEMCON-2003)*, Bhubaneswar, December 19-22, 2003.
10. Angira, R. and Babu, B. V., "Evolutionary Computation for Global Optimization of Non-Linear Chemical Engineering Processes". *Proceedings of International Symposium on Process Systems Engineering and Control (ISPSEC '03)- For Productivity Enhancement through Design and Optimization*, IIT-Bombay, Mumbai, January 3-4, 2003, Paper No. FMA2, pp 87-91.
11. Babu, B. V. and Angira, R., "Optimization Using Hybrid Differential Evolution Algorithms". *Proceedings of International Symposium & 57th Annual Session of IChE (CHEMCON-2004)*, Mumbai, December 27-30, 2004.
12. Angira, R. and Babu, B. V., "Optimization of Non-Linear Chemical Processes Using Modified Differential Evolution (MDE)", To be presented at The 2<sup>nd</sup> Indian International Conference on Artificial Intelligence (IICAI-2005), Pune, India, December 20-22, 2005.
13. Angira, R. and Babu, B. V., "Non-dominated Sorting Differential Evolution (NSDE): An Extension of Differential Evolution for Multi-objective Optimization", To be presented at The 2<sup>nd</sup> Indian International Conference on Artificial Intelligence (IICAI-2005), Pune, India, December 20-22, 2005.
14. Angira, R. and Babu, B. V., "Process Synthesis and Design Using Modified Differential Evolution (MDE)", To be presented at *International Symposium & 58th Annual Session of IChE* in association with International Partners (CHEMCON-2005), New Delhi, December 14-17, 2005.

### Biography of Guide

Dr. B. V. Babu is Professor & Head of Chemical Engineering Department apart from being Assistant Dean of Engineering Services Division (ESD) at Birla Institute of Technology and Science (BITS), Pilani. He is also the Programme Coordinator for the Off-Campus Programmes on Marine Engineering and Nautical Sciences, and BS (Process Engineering) of Distance Learning Programmes Division (DLPD) at BITS-Pilani. He did his PhD from IIT-Bombay. His biography is included in 2005 & 2006 editions of Marquis Who's Who in the World, and Thirty-Third Edition of the Dictionary of International Biography in September 2006.

He has 20 years of Teaching, Research, Consultancy, and Administrative experience. He guided 2 PhD students, 25 ME Dissertation students and 24 Thesis students and around 160 Project students. He is currently guiding 7 PhD candidates, 3 Dissertation students and 10 Project students. He currently has 3 research projects from UGC & DST.

His research interests include Evolutionary Computation (Population-based search algorithms for optimization of highly complex and non-linear engineering problems), Environmental Engineering, Biomass Gasification, Energy Integration, Artificial Neural Networks, Nano Technology, and Modeling & Simulation.

He is the recipient of National Technology Day (11<sup>th</sup> May, 2003) Award given by CSIR, obtained in recognition of the research work done in the area of 'A New Concept in Differential Evolution (DE) – Nested DE'. He is the Life member of Indian Institute of Chemical Engineers (IChE), Life member of Indian Society for Technical Education (ISTE), Life member of Institution of Engineers (IE), Fellow of International Congress of Chemistry and Environment (ICCE), Life member of Indian Environmental Association (IEA), Life member of Society of Operations Management (SOM), and Associate Member of International Society for Structural and Multidisciplinary Optimization (ISSMO). Nine of his technical papers have been included as successful applications of Differential Evolution (a populations based search algorithm) on the Homepage of Differential Evolution (<http://www.icsi.berkeley.edu/~storn/code.html#appl>).

He has around 102 research publications (International & National Journals & Conference Proceedings) to his credit. He completed three consultancy projects successfully and currently he is a Technical Consultant for Maharashtra Electricity Regulatory Commission (MERC), Mumbai and offering Advisory Services in the "Study relating to Bagasse Based Co-generation". He also has been invited as a consultant by a Bahrain (Middle East) based company for making a complex of chemical factories. He is a Panel Expert for [www.chemicalhouse.com](http://www.chemicalhouse.com) the most vibrant and active site for the Chemical Industry on the net which specializes in exchange of Information in a structured way among the chemical world in more than a Hundred Countries and a Million core chemical manufacturers traders scientists etc.



He has published four books (1) "Process Plant Simulation", EDD, BITS-Pilani, 2002, (2) "New Optimization Techniques in Engineering", Springer-Verlag, Germany, 2004, and (3) "Process Plant Simulation", Oxford University Press, India, 2004, (4) Energy Management Systems, EDD, BITS-Pilani, 2005. In addition he has written several chapters in various books and lecture notes of different intensive courses.

He was the Invited Chief Guest and delivered the Keynote addresses at one international conference (Desert Technology - 7) and three national seminars. He organized many Seminars & Conferences, and member of various academic and administrative committees at BITS-Pilani. He also chaired 10 Technical Sessions at various International & National Conferences. He delivered 23 invited lectures at various IITs and Universities. He is the Coordinator for PETROTECH Society at BITS-Pilani.

He is Editorial Board Member of two International Journals 'Energy Education Science & Technology' and 'Research Journal of Chemistry and Environment'. He is the referee & expert reviewer of 15 International Journals, and on the Programme Committees as an expert reviewer at 12 International Conferences (SEAL-2002 at Singapore; GECCO-2003 at Chicago, USA; CIRAS-2003 at Singapore; GECCO-2004 at Seattle, USA; SCI-2004 at Florida, USA; CCCT-2004 at Texas, USA; CIS-2004 at Singapore; ICARA-2004 at New Zealand; GECCO-2005 at Washington DC, USA; MEI-2005 at Austin, USA; Heat-SET-2005 at Grenoble, France; WMSCI-2005 at Florida, USA). He reviewed three books of McGraw Hill, Oxford University Press, and Tata McGraw Hill publishers. He is PhD Examiner for one candidate and PhD Thesis Reviewer for 3 Candidates.

He is also the Organizing Committee Member (Publicity Chair), and Session Organizer for the Special Session on "Evolutionary Computation" at The Second International Conference on "Computational Intelligence, Robotics, and Autonomous Systems (CIRAS-2003)", National University of Singapore, Singapore, December 15-17, 2003. He is the Session Chair and organized an Invited Session on "Engineering Applications of Evolutionary Computation Techniques" at "The Eighth World Multi-Conference on Systemics, Cybernetics, and Informatics (SCI-2004)", Orlando, Florida, USA, July 18-21, 2004.

### **Biography of Candidate**

Rakesh Angira completed his B. E. in Chemical engineering from C. R. State College of Engineering, Murthal, Sonapat (Haryana), India in 1994. After working for four years in industries, he joined academics and completed his M. E. in Chemical Engineering in 1999 from Birla Institute of Technology & Science (BITS), Pilani, India. Since then he joined as a faculty in Chemical Engineering Group, BITS, Pilani.

His research interests apart from Evolutionary Computation include simulation & optimization, process synthesis and design, optimal control, multi-objective optimization, and membrane separation processes. He has published one paper in International Journal of Computers & Chemical Engineering and eleven International and national conference papers. Also, two papers have been communicated to International Journals. He has been a program coordinator for B. S. (Process Engineering) program of DLP division. He is Associate member of Indian Institute of Chemical Engineers (IChE). He is co-author of the course material prepared for Seminar cum Intensive Course (SIC-2001) on Novel separation techniques and their application to chemical industries held at BITS, Pilani during March 21 – 23, 2001.

Code of Himmelblau function of Chapter – 2 is given in this Appendix.

## Code of Himmelblau function using DE and ten strategies (Chapter-2)

```

#define NP maxpop
#define D maxdim
#define gen_max genmax
#define F factor
#define CR cross
#define inibound_l inib_l
#define inibound_h inib_h
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#include<memory.h>
int strategy,genmax,maxdim,maxpop;
float inib_l,inib_h,factor,cross,cost[50],x1[50][10],x2[50][10];
float rnd_uni(long *);
void assignd(int D,float a[], float b[]);
float evaluate(float [],long *);
float evaluate(float tmp[],long *nfe)
{
    float cost;
    (*nfe)++;
    cost=pow((pow(tmp[0],2)+tmp[1]-1),2)+pow((tmp[0]+pow(tmp[1],2)-7),2);
    return cost;
}
float rnd_uni(long *idum)
{
    long j; long k;
    static long idum2=123456789;
    static long iy=0;static long iv[NTAB]; float temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;

```

```

        *idum=IA1*( *idum-k*IQ1)-k*IR1;
        if(*idum<0) *idum+=IM1;
        if(j<NTAB) iv[j]=*idum;
    }
    iy=iv[0];
}
k=( *idum)/IQ1;
*idum=IA1*( *idum-k*IQ1)-k*IR1;
if(*idum<0) *idum+=IM1;
k=idum2/IQ2;
idum2=IA2*(idum2-k*IQ2)-k*IR2;
if(idum2<0) idum2+=IM2;
j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
if(iy<1) iy+=IMM1;
if((temp=AM*iy)>RNMX) return RNMX;
else return temp;
}
}
void assignd(int D,float a[], float b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}
void main(int argc, char *argv[])
{
    int i,j,k,a,b,c,d,e,count=0,imin,seed;
    long nfe;
    float cost_trial,trial[3],costmin,bestit[3],best[3],cmax;
    clock_t start, end;
    FILE *fpin_ptr;
    if(argc!=2)
    {
        printf("\n Usage: De<input-file>\n");
        exit(1);
    }
    /*-----Read input data-----*/
    fpin_ptr = fopen(argv[1], "r");
    if(fpin_ptr==NULL)
    {
        printf("\n Cannot open input file\n");
        exit(1);
    }
    fscanf(fpin_ptr,"%d",&strategy);
    fscanf(fpin_ptr,"%d",&genmax);
    fscanf(fpin_ptr,"%d",&maxdim);
    fscanf(fpin_ptr,"%d",&maxpop);
    fscanf(fpin_ptr,"%f",&inib_l);
    fscanf(fpin_ptr,"%f",&inib_h);
    fscanf(fpin_ptr,"%f",&factor);
    fscanf(fpin_ptr,"%f",&cross);
    fscanf(fpin_ptr,"%d",&seed);
    fclose(fpin_ptr);
    long rnd_uni_init= -(long)seed; nfe=0;
    start = clock();
    for (i=0;i<NP;i++)
    {
        for (j=0;j<D;j++) /* rand()/32768.0*/

            x1[i][j]=inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h-inibound_l);
        cost[i]= evaluate(x1[i], &nfe);

        /* printf("x1=%f x2=%f cost=%f ",x1[i][0],x1[i][1],cost[i]);
        getch();*/
    }
    costmin=cost[0];

```

```

        imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]<costmin)
    {
        costmin=cost[i];
        imin=i;
    }
}

assignd(D,best,x1[imin]);
assignd(D,bestit,x1[imin]);
/*printf("\nbest=%f\n",best);*/
while (count<gen_max)
{
    count++;
    imin=0;
    for (i=0;i<NP;i++)
    {
        do a=int (rnd_uni(&rnd_uni_init)*NP); while (a==i);
        /*printf("a=%d ",a);*/
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        /*printf("\nb=%d ",b);*/
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        /*printf("\n c=%d",c); */
        do d=int (rnd_uni(&rnd_uni_init)*NP); while (d==i || d==a || d==b || d==c);
        do e=int (rnd_uni(&rnd_uni_init)*NP); while (e==i || e==a || e==b || e==c || e==d);
        /*-----de/rand/1/bin-----*/
        if (strategy==1)
        {
            j=int (rnd_uni(&rnd_uni_init)*D);
            /*printf(" j=%d" j);
            getch(); */

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

                /*printf("r1=%f ,trial[%d]=%f , ",r1,j,trial[j]);
                getch();*/
                j=(j+1)%D;
            }
        }
        /*-----DE/best/1/bin-----*/
    else if (strategy==2)
    {
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if ((rnd_uni(&rnd_uni_init))<CR || k==D)
            {
                trial[j]=bestit[j]+F*(x1[a][j]-x1[b][j]);
            }
            else trial[j]=x1[i][j];

            j=(j+1)%D;
        }
    }
    /*-----de/best/2/bin-----*/
    else if (strategy==3)
    {
        assignd(D,trial,x1[i]);
        j=int (rnd_uni(&rnd_uni_init)*D);

```

```

        for (k=1;k<=D;k++)
        {
            if ((rnd_uni(&rnd_uni_init))<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];
            j=(j+1)%D;
        }
    }
    /*-----de/rand/2/bin-----*/
    else if (strategy==4)
    {
        assignd(D,trial,x1[i]);
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if ((rnd_uni(&rnd_uni_init))<CR || k==D)
            {
                trial[j]=x1[c][j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);
            }
            else trial[j]=x1[i][j];
            j=(j+1)%D;
        }
    }
    /*-----de/rand-to-best/1/bin-----*/
    else if (strategy==5)
    {
        assignd(D,trial,x1[i]);
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if ((rnd_uni(&rnd_uni_init))<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];
            j=(j+1)%D;
        }
    }
    /*-----de/rand/1/exp-----*/
    else if (strategy==6)
    {
        j=int (rnd_uni(&rnd_uni_init)*D);
        k=0;
        do
        {
            trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);

            j=(j+1)%D;
            k++;
        }
        while((rnd_uni(&rnd_uni_init))<CR && k<D);
    }
    /*-----de/best/1/exp-----*/
    else if (strategy==7)
    {
        j=int (rnd_uni(&rnd_uni_init)*D);
        k=0;
        do
        {
            trial[j]=bestit[j]+F*(x1[a][j]-x1[b][j]);

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

```

```

}

/*-----de/best/2/exp-----*/
else if (strategy==8)
{
    assignd(D,trial,x1[i]);
    j=int (rnd_uni(&rnd_uni_init)*D);
    k=0;
    do
    {
        trial[j]=bestit[j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);

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

        j=(j+1)%D;
        k++;
    }
    while((rnd_uni(&rnd_uni_init))<CR && k<D);
}
/*-----de/rand-to-best/1/exp-----*/
else
{
    assignd(D,trial,x1[i]);
    j=int (rnd_uni(&rnd_uni_init)*D);
    k=0;
    do
    {
        trial[j]=trial[j]+F*(bestit[j]-trial[j])+F*(x1[a][j]-x1[b][j]);

        j=(j+1)%D;
        k++;
    }
    while((rnd_uni(&rnd_uni_init))<CR && k<D);
}
cost_trial=evaluate(trial, &nfe);
/* printf("\ntrialcost=%f , cost[%d]=%f ",cost_trial,i,cost[i]);
getch();*/
if (cost_trial<=cost[i])
{
    for (j=0;j<D;j++)
        x2[i][j]=trial[j];
    cost[i]=cost_trial;
    if(cost_trial<costmin)
    {
        costmin=cost_trial;
        imin=i;
        assignd(D,best,trial);
    }
}
else for (j=0;j<D;j++)
    x2[i][j]=x1[i][j];
/* printf("x1=%f x2=%f ",x2[i][0],x2[i][1]);
getch(); */
} /*-----end of FOR loop after while-----*/

```

```

    assignd(D,bestit,best);
    for (i=0;i<NP;i++)
    {
        for (j=0;j<D;j++)
            x1[i][j]=x2[i][j];
    }
    cmax=cost[0];
    for (i=1;i<NP;i++)
    {
        if(cost[i]>cmax)
            cmax=cost[i];
    }
    if((cmax-0.0)<=0.000001)
        break;
} /*-----end of while loop-----*/
for(i=0;i<NP;i++)
{
    printf("x1=%f x2=%f ",x1[i][0],x1[i][1]);
    printf("cost[%d]=%f ",i,cost[i]);
}
printf("\ncmax=%f\n",cmax);
printf("\ncount=%d\n",count);
printf("\ncostmin=%f\n",costmin);
printf("\n NFE=%ld\n",nfe);
end = clock();
printf("The time was: %f\n", (end - start) / CLK_TCK);
} /*-----end of main()-----*/
/* while (count<gen_max)
for (i=0;i<NP;i++)
{
    do a=rdm_uni()*NP; while (a==i);
    do b=rdm_uni()*NP; while (b==i || b==a);
    do c=rdm_uni()*NP; while (c==i || c==a || c==b);
    j=rdm_uni()*D;
    for (k=1;k<=D;k++)
    {
        if (rdm_uni() < CR || k==D)
        {
            trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
        }
        else trial[j]=x1[i][j];
        j=(j+1)/D;
    }
    score=evaluate(trial);
    if (score<=cost[i])
    {
        for (j=0;j<D;j++)
            x2[i][j]=trial[j];
        cost[i]=score;
    }
    else for (j=0;j<D;j++)
        x2[i][j]=x1[i][j];
}

for (i=0;i<NP;i++)
{
    for (j=0;j<D;j++)
        x1[i][j]=x2[i][j];
}
count++;
} */

```

---

## APPENDIX II

---

Code of selected problems from Chapter – 3 are given in this Appendix.

### Code of Himmelblau function using MDE (Chapter-3)

```
#define gen_max 1000
#define D 2
#define NP D*10
#define F 0.8          /* 0.64 & 0.51*/
#define CR 0.5
#define Lo 0.0
#define Hi 6.0
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#include<memory.h>
float evaluate(double [], long *);
float evaluate(double tmp[], long *nfe)
    { double z,z1,z2,x1,x2; /*** x1=tmp[0], x2=tmp[1]**/
      (*nfe)++;

      x1=tmp[0];x2=tmp[1];
      z=(pow(((x1*x1)+x2-11),2))+pow((x1+(x2*x2)-7),2));
      return z;
    }
float rnd_uni(long *);
float rnd_uni(long *idum)
    { long j; long k;
      static long idum2=123456789;
      static long iy=0;static long iv[NTAB]; float temp;
      if(*idum<=0)
        {
          if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
          for(j=NTAB+7;j>=0;j--)
            {
              k=(*idum)/IQ1;
              *idum=IA1*( *idum-k*IQ1)-k*IR1;
              if(*idum<0) *idum+=IM1;
              if(j<NTAB) iv[j]=*idum;
            }
        }
    }
```



```

    }
    costmax=cost[0];
    for(i=1;i<NP;i++)
    { if(costmax<cost[i])
      costmax=cost[i];
    }

    count++;
    if(fabs(costmin-(0.0))<1e-6)
      break;
} /****end of while loop*****/

end = clock();

for(j=0;j<D;j++)
  printf(" x1=%f ",x1[mini][j]);
  printf("cost[%d]=%f ",mini,cost[mini]);

  printf("NFE=%ld genmax=%d\n",nfe,count);
  printf("The time was: %f\n", (end - start) / CLK_TCK);
  printf("cost=%f",evaluate(x1[0],&nfe));
  printf("costmin=%f",costmin);
  printf("costmax=%f",costmax);
} /*****end of main() *****/

```

### Code of thermal cracker problem using DE (Chapter-3)

```

#define gen_max 1500
#define D 4
#define NP 40
#define F 0.5 /* F=0.5 & CR=0.8*/
#define CR 0.8
#define inibound_l 0
#define inibound_h 90000
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#include<memory.h>
void assignd(double a[], double b[]);
void assignd(double a[], double b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}

```

```

    }
    costmax=cost[0];
    for(i=1;i<NP;i++)
    { if(costmax<cost[i])
      costmax=cost[i];
    }

    count++;
    if(fabs(costmin-(0.0))<1e-6)
      break;
} /****end of while loop*****/

end = clock();

for(j=0;j<D;j++)
  printf(" x1=%f ",x1[mini][j]);
  printf("cost[%d]=%f ",mini,cost[mini]);

  printf("NFE=%ld genmax=%d\n",nfe,count);
  printf("The time was: %f\n", (end - start) / CLK_TCK);
  printf("cost=%f",evaluate(x1[0],&nfe));
  printf("costmin=%f",costmin);
  printf("costmax=%f",costmax);
} /****end of main() *****/

```

### Code of thermal cracker problem using DE (Chapter-3)

```

#define gen_max 1500
#define D 4
#define NP 40
#define F 0.5 /* F=0.5 & CR=0.8*/
#define CR 0.8
#define inibound_l 0
#define inibound_h 90000
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#include<memory.h>
void assignd(double a[], double b[]);
void assignd(double a[], double b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}

```

```

lhs3=(0.15*x1[i][0])+(1.51*x1[i][1])+(1.3711*x1[i][2])+(1.6426*x1[i][3]);
if(lhs3>180000.0)
{
    pen=(lhs3*100);
    cost[i]=evaluate(x1[i],pen,&nfc);
    continue;
}

if(lhs1<=1800000 && lhs2<=450000.0 && lhs3<=180000.0)
{
    cost[i]=evaluate(x1[i],pen,&nfc);
}
}
costmax=cost[0];
imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]>costmax)
    {
        costmax=cost[i];
        imin=i;
    }
}
assignd(best,x1[imin]);
assignd(bestit,x1[imin]);
while (count<gen_max)
{
    for (i=0;i<NP;i++)
    {
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        do d=int (rnd_uni(&rnd_uni_init)*NP); while (d==i || d==a || d==b || d==c);
        do e=int (rnd_uni(&rnd_uni_init)*NP); while (e==i || e==a || e==b || e==c || e==d);
        /*-----de/rand/1/bin-----*/
        if (strategy==1)
        {
            j=int (rnd_uni(&rnd_uni_init)*D);

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

                if(trial[j]<0.0)          trial[j]=0.0;

                j=(j+1)%D;
            }
        }
        /*-----DE/best/1/bin-----*/
        else if (strategy==2)
        {
            j=int (rnd_uni(&rnd_uni_init)*D);

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

```

```

lhs3=(0.15*x1[i][0])+(1.51*x1[i][1])+(1.3711*x1[i][2])+(1.6426*x1[i][3]);
if(lhs3>180000.0)
{
    pen=(lhs3*100);
    cost[i]=evaluate(x1[i],pen,&nfe);
    continue;
}

if(lhs1<=1800000 && lhs2<=450000.0 && lhs3<=180000.0)
{
    cost[i]=evaluate(x1[i],pen,&nfe);
}
}
costmax=cost[0];
imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]>costmax)
    {
        costmax=cost[i];
        imin=i;
    }
}
assignd(best,x1[imin]);
assignd(bestit,x1[imin]);
while (count<gen_max)
{
    for (i=0;i<NP;i++)
    {
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        do d=int (rnd_uni(&rnd_uni_init)*NP); while (d==i || d==a || d==b || d==c);
        do e=int (rnd_uni(&rnd_uni_init)*NP); while (e==i || e==a || e==b || e==c || e==d);
        /*-----de/rand/1/bin-----*/
        if (strategy==1)
        {
            j=int (rnd_uni(&rnd_uni_init)*D);

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

                if(trial[j]<0.0)          trial[j]=0.0;

                j=(j+1)%D;
            }
        }
        /*-----DE/best/1/bin-----*/
        else if (strategy==2)
        {
            j=int (rnd_uni(&rnd_uni_init)*D);

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

```

```

else if (strategy==6)
    {
        j=int (rnd_uni(&rnd_uni_init)*D);
        k=0;
        do
            {
                trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
                if(trial[j]<0.0)          trial[j]=0.0;
                j=(j+1)%D;
                k++;
            }
            while((rnd_uni(&rnd_uni_init)<CR && k<D);
    }

/*-----de/best/1/exp-----*/

else if (strategy==7)
    {
        j=int (rnd_uni(&rnd_uni_init)*D);
        k=0;
        do
            {
                trial[j]=bestit[j]+F*(x1[a][j]-x1[b][j]);
                if(trial[j]<0.0)          trial[j]=0.0;
                j=(j+1)%D;
                k++;
            }
            while((rnd_uni(&rnd_uni_init)<CR && k<D);
    }

/*-----de/best/2/exp-----*/

else if (strategy==8)
    {
        assignd(trial,x1[i]);
        j=int (rnd_uni(&rnd_uni_init)*D);
        k=0;
        do
            {
                trial[j]=bestit[j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);
                if(trial[j]<0.0)          trial[j]=0.0;
                j=(j+1)%D;
                k++;
            }
            while((rnd_uni(&rnd_uni_init)<CR && k<D);
    }

/*-----de/rand/2/exp-----*/

else if (strategy==9)
    {
        assignd(trial,x1[i]);
        j=int (rnd_uni(&rnd_uni_init)*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]<0.0)          trial[j]=0.0;
                j=(j+1)%D;
                k++;
            }
            while((rnd_uni(&rnd_uni_init)<CR && k<D);
    }

/*-----de/rand-to-best/1/exp-----*/

```

```

else if (strategy==6)
{
    j=int (rnd_uni(&rnd_uni_init)*D);
    k=0;
    do
    {
        trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
        if(trial[j]<0.0) trial[j]=0.0;
        j=(j+1)%D;
        k++;
    }
    while((rnd_uni(&rnd_uni_init))<CR && k<D);
}

/*-----de/best/1/exp-----*/

else if (strategy==7)
{
    j=int (rnd_uni(&rnd_uni_init)*D);
    k=0;
    do
    {
        trial[j]=bestit[j]+F*(x1[a][j]-x1[b][j]);
        if(trial[j]<0.0) trial[j]=0.0;
        j=(j+1)%D;
        k++;
    }
    while((rnd_uni(&rnd_uni_init))<CR && k<D);
}

/*-----de/best/2/exp-----*/

else if (strategy==8)
{
    assignd(trial,x1[i]);
    j=int (rnd_uni(&rnd_uni_init)*D);
    k=0;
    do
    {
        trial[j]=bestit[j]+F*(x1[a][j]+x1[b][j]-x1[c][j]-x1[d][j]);
        if(trial[j]<0.0) trial[j]=0.0;
        j=(j+1)%D;
        k++;
    }
    while((rnd_uni(&rnd_uni_init))<CR && k<D);
}

/*-----de/rand/2/exp-----*/

else if (strategy==9)
{
    assignd(trial,x1[i]);
    j=int (rnd_uni(&rnd_uni_init)*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]<0.0) trial[j]=0.0;
        j=(j+1)%D;
        k++;
    }
    while((rnd_uni(&rnd_uni_init))<CR && k<D);
}

/*-----de/rand-to-best/1/exp-----*/

```

```

        cost_trial=evaluate(trial,pen,&nfe);
        if (cost_trial>=cost[i])
            {
                for (j=0;j<D;j++)
                    x2[i][j]=trial[j];
                cost[i]=cost_trial;
                if(cost_trial>costmax)
                    {
                        costmax=cost_trial;
                        imin=i;
                        assignd(best,trial);
                    }
            }
        else for (j=0;j<D;j++)
            x2[i][j]=x1[i][j];
        continue;
    }

if((lhs1<=1800000 && lhs2<=450000.0 && lhs3<=(180000.0))
    {
        cost_trial=evaluate(trial,pen,&nfe);

        if(cost_trial>=cost[i])
            {
                for (j=0;j<D;j++)
                    x2[i][j]=trial[j];
                cost[i]=cost_trial;
                if(cost_trial>costmax)
                    {
                        costmax=cost_trial;
                        imin=i;
                        assignd(best,trial);
                    }
            }
        else for (j=0;j<D;j++)
            x2[i][j]=x1[i][j];
        continue;
    }
} /*****end of for loop*****/
assignd(bestit,best);
for(i=0;i<NP;i++)
    {
        for (j=0;j<D;j++)
            x1[i][j]=x2[i][j];
    }

    costmax=cost[0];
    for(i=1;i<NP;i++)
        { if(costmax<cost[i])
            costmax=cost[i];
        }
    costmin=cost[0];
    for(i=1;i<NP;i++)
        { if(costmin>cost[i])
            costmin=cost[i];
        }

    if((costmax-costmin)<0.000001)
        break;

    count++;
} /*****end of while loop*****/

end = clock();

for(i=0;i<NP;i++)

```

```

        cost_trial=evaluate(trial,pen,&nfc);
        if (cost_trial>=cost[i])
            {
                for (j=0;j<D;j++)
                    x2[i][j]=trial[j];
                cost[i]=cost_trial;
                if(cost_trial>costmax)
                    {
                        costmax=cost_trial;
                        imin=i;
                        assignd(best,trial);
                    }
            }
        else for (j=0;j<D;j++)
            x2[i][j]=x1[i][j];
        continue;
    }

if(lhs1<=1800000 && lhs2<=450000.0 && lhs3<=(180000.0))
    {
        cost_trial=evaluate(trial,pen,&nfc);

        if(cost_trial>=cost[i])
            {
                for (j=0;j<D;j++)
                    x2[i][j]=trial[j];
                cost[i]=cost_trial;
                if(cost_trial>costmax)
                    {
                        costmax=cost_trial;
                        imin=i;
                        assignd(best,trial);
                    }
            }
        else for (j=0;j<D;j++)
            x2[i][j]=x1[i][j];
        continue;
    }
} /*****end of for loop*****/
assignd(bestit,best);
for(i=0;i<NP;i++)
    {
        for (j=0;j<D;j++)
            x1[i][j]=x2[i][j];
    }

    costmax=cost[0];
    for(i=1;i<NP;i++)
        { if(costmax<cost[i])
            costmax=cost[i];
        }
    costmin=cost[0];
    for(i=1;i<NP;i++)
        { if(costmin>cost[i])
            costmin=cost[i];
        }

    if((costmax-costmin)<0.000001)
        break;

    count++;
} /*****end of while loop*****/

end = clock();

for(i=0;i<NP;i++)

```



```

    {
        printf("x1=%f x2=%f x3=%f x4=%f ",x1[i][0],x1[i][1],x1[i][2],x1[i][3]);
        printf("cost[%d]=%f ",i,cost[i]);
    }

printf("lhs1=%f lhs2=%f lhs3=%f \n ",lhs1,lhs2,lhs3);
printf("NFE=%ld\n",nfe);
printf("The time was: %f\n", (end - start) / CLK_TCK);
printf("would you like to exit? press Y for exit");
ch=getch();
if(ch=='y'||ch=='Y') { printf("exited"); exit(1);}

fout=fopen("\\out100.xls","a+");
/* fprintf(fout,"n\nThe out put is\n:");
for(i=0;i<NP;i++)
    { if(i%10==0)
        {
            fprintf(fout, "x1=%f x2=%f x3=%f x4=%f ",x1[i][0],x1[i][1],x1[i][2],x1[i][3]);
            fprintf(fout, "cost[%d]=%f\n",i,cost[i]);
        }
    }
*/
fprintf(fout, "%d %ld ",strategy,nfe);
fprintf(fout, "%f ", (end - start) / CLK_TCK);
fprintf(fout, "%d %d %f %f\n",count,seed,F,CR);
fclose(fout);
} /***** end of main() *****/

```

### Code of thermal cracker problem using MDE (Chapter – 3)

```

#define gen_max 5000
#define D 4
#define NP 40
#define F 0.8          /* F=0.5 & CR=0.8*/
#define CR 0.5
#define inibound_l 0
#define inibound_h 90000
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#include<memory.h>
void assignd(double a[], double b[]):
void assignd(double a[], double b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}

```

```

float evaluate(double [], long *);
float evaluate(double tmp[], long *nfe)
{
    double cost,lhs1,lhs2,lhs3; (*nfe)++;

    lhs1=(16.5*tmp[0])+(10.1*tmp[1])+(8.861*tmp[2])+(9.926*tmp[3]);
    lhs2=(7.5*tmp[0])+(4.0*tmp[1])+(2.14*tmp[2])+(2.665*tmp[3]);
    lhs3=(0.15*tmp[0])+(1.51*tmp[1])+(1.3711*tmp[2])+(1.6426*tmp[3]);
    if(lhs1<=1800000 && lhs2<=450000.0 && lhs3<=180000.0)
    {
        cost=(tmp[0]*9.1)+(tmp[1]*16.92/9.0)-(tmp[2]*23.2911/9)+(tmp[3]*17.8974/9);
    }
    else cost=(tmp[0]*9.1)+(tmp[1]*16.92/9.0)-(tmp[2]*23.2911/9)+(tmp[3]*17.8974/9)-(lhs1+lhs2+lhs3)*1e6;

    return cost;
}

float rnd_uni(long *);

float rnd_uni(long *idum)
{
    long j; long k;
    static long idum2=123456789;
    static long iy=0;static long iv[NTAB]; float temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;
            *idum=IA1*(*idum-k*IQ1)-k*IR1;
            if(*idum<0) *idum+=IM1;
            if(j<NTAB) iv[j]=*idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*(*idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
    j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
    if(iy<1) iy+=IMM1;
    if((temp=AM*iy)>RNMXX) return RNMXX;
    else return temp;
}

void main()

{
    int i,j,k,a,b,c,d,e,good,count,seed,imin; long nfe;
    double x1[NP][D],cost[NP],trial[D];
    double cost_trial,costmin,costmax,bestit[D],best[D];
    clock_t start, end; FILE *fout; char ch;

    // printf("\nseed="); scanf("%d",&seed);
    // printf("\nstrategy="); scanf("%d",&strategy);
    for(d=1;d<=100;d++)
    {
        seed=rand()%9999; nfe=0; count=0;
        long rnd_uni_init=-(long)seed; start = clock();

        for (i=0;i<NP;i++)
        {
            for (j=0;j<D;j++)
            {
                x1[i][j]=inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h-inibound_l);
            }
        }
    }
}

```

```

    cost[i]=evaluate(x1[i],&nfc);
}
costmax=cost[0];
imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]>costmax)
    {
        costmax=cost[i];
        imin=i;
    }
}
assignd(best,x1[imin]);
assignd(bestit,x1[imin]);
while (count<gen_max)
{
    for (i=0;i<NP;i++)
    {
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        // do d=int (rnd_uni(&rnd_uni_init)*NP); while (d==i || d==a || d==b || d==c);
        // do e=int (rnd_uni(&rnd_uni_init)*NP); while (e==i || e==a || e==b || e==c || e==d);
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if ((rnd_uni(&rnd_uni_init))<CR || k==D)
            {
                trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
            }
            else trial[j]=x1[i][j];

            if(trial[j]<0.0) trial[j]=0.0+rnd_uni(&rnd_uni_init)*90000.0;

            j=(j+1)%D;
        }
        cost_trial=evaluate(trial,&nfc);

        if(cost_trial>=cost[i])
        {
            for (j=0;j<D;j++)
                x1[i][j]=trial[j];
            cost[i]=cost_trial;
            if(cost_trial>costmax)
            {
                costmax=cost_trial;
                imin=i;
                assignd(best,trial);
            }
        }
        // else for (j=0;j<D;j++)
        //     x2[i][j]=x1[i][j];
    }
} /*****end of for loop*****/
assignd(bestit,best);
/* for(i=0;i<NP;i++)
{
    for (j=0;j<D;j++)
        x1[i][j]=x2[i][j];
}
*/
costmax=cost[0];
for(i=1;i<NP;i++)
{ if(costmax<cost[i])
    costmax=cost[i];
}

```

```

        costmin=cost[0];
    for(i=1;i<NP;i++)
        { if(costmin>cost[i])
            costmin=cost[i];
        }
        count++;

    if((costmax-costmin)<0.000001)
        break;

; /*****end of while loop*****/
end = clock();
for(i=0;i<NP;i++)
    {
        printf("x1=%f x2=%f x3=%f x4=%f ",x1[i][0],x1[i][1],x1[i][2],x1[i][3]);
        printf("cost[%d]=%f ",i,cost[i]);
    }

// printf("lhs1=%f lhs2=%f lhs3=%f \n ",lhs1,lhs2,lhs3);
printf("NFE=%ld\n",nfe);
printf("The time was: %f\n", (end - start) / CLK_TCK);
printf("would you like to exit? press Y for exit");
ch=getch();
if(ch=='y'||ch=='Y') { printf("exited"); exit(1);}

fout=fopen("\\outMDE10.xls","a+");
/* fprintf(fout,"The out put is\n");
for(i=0;i<NP;i++)
    { if(i%10==0)
        {
            fprintf(fout, "x1=%f x2=%f x3=%f x4=%f ",x1[i][0],x1[i][1],x1[i][2],x1[i][3]);
            fprintf(fout, "cost[%d]=%f\n",i,cost[i]);
        }
    }
*/
fprintf(fout, "%ld ",nfe);
fprintf(fout, "%f ", (end - start) / CLK_TCK);
fprintf(fout, "%d %d %f %f\n",count,seed,F,CR);
fclose(fout);
} /***** end of main() *****/
}

```

### Code of Runge Kutta method for Ammonia Synthesis problem (Chapter – 3)

```

#include<stdlib.h>
#include<stdio.h>          /* Please check the f1 & f2 values*/
#include<math.h>
#include<conio.h>

#define f1 1.78954e4
#define f2 2.5714e16
#define R 1.987
#define hh 0.0001
#define N2o 701.2
#define To 694.0
#define Cpf 0.707
#define Cpg 0.719
#define dH 26600.0
#define E1 20800.0
#define E2 47400.0
#define S1 10.0
#define S2 0.78
#define U 500.0
#define W 26400.0

double RKM(double rl);
FILE* galog;

```

```

void main()
{ double rl=10.0,profit;

profit=RKM(rl);    printf("profit=%lf\n",profit);
}
double RKM(double rl)
{
double x,h,Tg,Tf,N2,Tfo,Tgo,cost,N20,pN2,pNH3,pH2,Tempg[1000],Tempf[1000];
double K1,K2,k1,k2,k3,k4,l1,l2,l3,l4,m1,m2,m3,m4,NN2[1000],L[1000];
int ij;  x=0.0; Tg=To; Tf=To; N2=N20=N2o; Tfo=Tgo=To; long nstep;

/*printf("Reactor Length=");
scanf("%lf",&rl);*/

nstep= long(rl/hh);
nstep= nstep+1;  printf("nstep=%ld",nstep); getch();
h=hh;
for(i=0;i<nstep;i++)
{
    k1=h*(-(U*S1*(Tg-Tf)/(W*Cpf)));
    K1=f1*exp(-E1/(R*Tg));
    K2=f2*exp(-E2/(R*Tg));
    pN2=(286.0*N2/(2.598*N2o+2.0*N2));          pH2=3.0*pN2;          pNH3=(286.0*(2.23*N2o-
    2.0*N2)/(2.598*N2o+2.0*N2));
    m1=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
    l1=h*(-(U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m1/h)/(W*Cpg)));

/* printf("k1=%f  m1=%f l1=%f\n",k1,m1,l1); getch(); */

Tg=Tg+(l1/2.0); Tf=Tf+(k1/2.0); N2=N2+(m1/2.0);

    k2=h*(-(U*S1*(Tg-Tf)/(W*Cpf)));
    K1=f1*exp(-E1/(R*Tg));
    K2=f2*exp(-E2/(R*Tg));
    pN2=(286.0*N2/(2.598*N2o+2.0*N2)); pH2=3*pN2; pNH3=(286.0*(2.23*N2o-2.0*N2)/(2.598*N2o+2.0*N2));
    m2=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
    l2=h*(-(U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m2/h)/(W*Cpg)));
/* printf("k2=%f  m2=%f  l2=%f  \n",k2,m2,l2); getch(); */
Tg=Tg+(l2/2.0); Tf=Tf+(k2/2.0); N2=N2+(m2/2.0);

    k3=h*(-(U*S1*(Tg-Tf)/(W*Cpf)));
    K1=f1*exp(-E1/(R*Tg));
    K2=f2*exp(-E2/(R*Tg));
    pN2=(286.0*N2/(2.598*N2o+2.0*N2)); pH2=3*pN2; pNH3=(286.0*(2.23*N2o-2.0*N2)/(2.598*N2o+2.0*N2));
    m3=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
    l3=h*(-(U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m3/h)/(W*Cpg)));

/* printf("k3=%f  m3=%f  l3=%f  \n",k3,m3,l3); */

Tg=Tg+l3; Tf=Tf+k3; N2=N2+m3;

    k4=h*(-(U*S1*(Tg-Tf)/(W*Cpf)));
    K1=f1*exp(-E1/(R*Tg));
    K2=f2*exp(-E2/(R*Tg));
    pN2=(286.0*N2/(2.598*N2o+2.0*N2)); pH2=3*pN2; pNH3=(286.0*(2.23*N2o-2.0*N2)/(2.598*N2o+2.0*N2));
    m4=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
/* m4=h*((-4.5*K1*286.0*N2*N2)/((2.598*N2o+2.0*N2)*(2.23*N2o-2.0*N2)))+(K2*(2.23*N2o-
    2*N2)/(4.5*N2));*/
    l4=h*(-(U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m4/h)/(W*Cpg)));

/* printf("k4=%f  m4=%f  l4=%f  \n",k4,m4,l4); */

Tf=Tfo+(k1+(2*k2)+(2*k3)+k4)/6.0;
Tg=Tgo+(l1+(2*l2)+(2*l3)+l4)/6.0;
N2=N20+(m1+(2*m2)+(2*m3)+m4)/6.0;
x=x+h;    if(Tf<400.0) break;
}
}

```

```

Tgo=Tg; Tfo=Tf; N2o=N2;
if ((galog = fopen("D:\\de2.xls","a+"))==NULL)
{
    exit(1);
}
fprintf(galog,"x=%lf      Tf=%lf  Tg=%lf  N2=%lf\n",x,Tfo,Tgo,N2o);
fclose(galog);
}

/*Tempg[j]=Tg; Tempf[j]=Tf; NN2[j]=N2; L[j]=x;

printf("x[%d]=%lf  Tf[%d]=%lf  Tg[%d]=%lf  N2[%d]=%lf\n",j,L[j],Tempf[j],Tempg[j],NN2[j]);
*/
cost=(13356300.0)-(17084.3*N2)+(704.09*(Tg-To))-(699.27*(Tf-To))-sqrt((34566300.0)+(1983650000.0*x));
printf("x=%lf , Tg=%f  Tf=%f  N2=%f\n",x,Tg,Tf,N2);
printf("cost=%lf\n",cost);
return(cost);
}

```

### Code of Ammonia Synthesis problem using DE (Chapter – 3)

```

#define gen_max 100
#define D 1
#define NP 10
#define F 0.5
#define CR 0.9 /* 0.6 */
#define inibound_l 0 /* Please Check f1 & f2 values*/
#define inibound_h 10

/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)

/*-----Parameter's Value for solving ODE's -----*/
#define f1 1.78954e4
#define f2 2.5714e16
#define R 1.987
#define hh 0.01
#define N2o 701.2
#define To 694.0
#define Cpf 0.707
#define Cpg 0.719
#define dH 26600.0
#define E1 20800.0
#define E2 47400.0
#define S1 10.0
#define S2 0.78
#define U 500.0
#define W 26400.0

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>

```

```

#include<conio.h>

static double Tg,Tf,N2,x;

void assignd(double a[], double b[]):

void assignd(double a[], double b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}

float rnd_uni(long *);

float rnd_uni(long *idum)
{
    long j; long k;
    static long idum2=123456789;
    static long iy=0;static long iv[NTAB]; float temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;
            *idum=IA1*( *idum-k*IQ1)-k*IR1;
            if(*idum<0) *idum+=IM1;
            if(j<NTAB) iv[j]=*idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*( *idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
    j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
    if(iy<1) iy+=IMM1;
    if((temp=AM*iy)>RNMX) return RNMX;
    else return temp;
}

double RKM(double rl);

double RKM(double rl)
{
    double h,Tfo,Tgo, cost,N20,pN2,pNH3,pH2; long nstep;
    double K1,K2,k1,k2,k3,k4,l1,l2,l3,l4,m1,m2,m3,m4;
    int ij; x=0.0; Tg=To; Tf=To; N2=N20=N2o; Tfo=Tgo=To;

    /*printf("Reactor Length=");
    scanf("%lf",&rl);*/

    nstep= long(rl/hh); /*printf("nstep=%ld",nstep); getch(); */

    h=hh;

    for(i=0;i<nstep;i++)
    {
        k1=h*(-(U*S1*(Tg-Tf)/(W*Cpf)));
        K1=f1*exp(-E1/(R*Tg));
        K2=f2*exp(-E2/(R*Tg));
    }
}

```

```

pN2=(286.0*N2/(2.598*N2o+2.0*N2));          pH2=3.0*pN2;          pNH3=(286.0*(2.23*N2o-
2.0*N2)/(2.598*N2o+2.0*N2));
    m1=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
    l1=h*((-U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m1/h)/(W*Cpg)));

/* printf("k1=%f m1=%f l1=%f\n",k1,m1,l1); getch(); */

Tg=Tg+(l1/2.0); Tf=Tf+(k1/2.0); N2=N2+(m1/2.0);

    k2=h*((-U*S1*(Tg-Tf)/(W*Cpf));
K1=f1*exp(-E1/(R*Tg));
K2=f2*exp(-E2/(R*Tg));
pN2=(286.0*N2/(2.598*N2o+2.0*N2)); pH2=3*pN2; pNH3=(286.0*(2.23*N2o-2.0*N2)/(2.598*N2o+2.0*N2));
    m2=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
    l2=h*((-U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m2/h)/(W*Cpg)));

/* printf("k2=%f m2=%f l2=%f\n",k2,m2,l2); getch(); */

Tg=Tg+(l2/2.0); Tf=Tf+(k2/2.0); N2=N2+(m2/2.0);

    k3=h*((-U*S1*(Tg-Tf)/(W*Cpf));
K1=f1*exp(-E1/(R*Tg));
K2=f2*exp(-E2/(R*Tg));
pN2=(286.0*N2/(2.598*N2o+2.0*N2)); pH2=3*pN2; pNH3=(286.0*(2.23*N2o-2.0*N2)/(2.598*N2o+2.0*N2));
    m3=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
    l3=h*((-U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m3/h)/(W*Cpg)));

/* printf("k3=%f m3=%f l3=%f\n",k3,m3,l3); */

Tg=Tg+l3; Tf=Tf+k3; N2=N2+m3;

    k4=h*((-U*S1*(Tg-Tf)/(W*Cpf));
K1=f1*exp(-E1/(R*Tg));
K2=f2*exp(-E2/(R*Tg));
pN2=(286.0*N2/(2.598*N2o+2.0*N2)); pH2=3*pN2; pNH3=(286.0*(2.23*N2o-2.0*N2)/(2.598*N2o+2.0*N2));
    m4=-h*((K1*pN2*pow(pH2,1.5)/pNH3)-(K2*pNH3/(pow(pH2,1.5))));
    /* m4=h*(((-4.5*K1*286.0*N2*N2)/((2.598*N2o+2.0*N2)*(2.23*N2o-2.0*N2)))+(K2*(2.23*N2o-
2*N2)/(4.5*N2));*/
    l4=h*((-U*S1*(Tg-Tf)/(W*Cpg))+(dH*S2*(-m4/h)/(W*Cpg)));

/* printf("k4=%f m4=%f l4=%f\n",k4,m4,l4); */

Tf=Tfo+(k1+(2*k2)+(2*k3)+k4)/6.0;
Tg=Tgo+(l1+(2*l2)+(2*l3)+l4)/6.0;
N2=N20+(m1+(2*m2)+(2*m3)+m4)/6.0;

x=x+h;          if( Tf<=400.0) break;

Tgo=Tg; Tfo=Tf; N20=N2;

}
cost=(13356300.0)-(17084.3*N2)+(704.09*(Tg-To))-(699.27*(Tf-To))-sqrt((34566300.0)+(1983650000.0*x));
return(cost);
/*printf("cost=%f\n",cost); */

}

void main()

{
int i,j,k,a,b,c,good,count=0,seed,imin; long nfe=0;
double x1[NP][D],x2[NP][D],cost[NP],trial[D],rl,best[D],bestit[D];
double cost_trial,costmax,costmin,pen; clock_t start, end;

printf("\nseed=");

```



```

scanf("%d",&seed);
long rnd_uni_init= -(long)seed; start = clock();

for (i=0;i<NP;i++)
{
    for (j=0;j<D;j++)
    {
        x1[i][j]=inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h-inibound_l);

    }
    /* printf("x=%f\n",x1[i][0]); */

    rl=x1[i][0];
    cost[i]=RKM(rl);    x1[i][0]=x;

    /* printf("cost[%d]=%f\n",i,cost[i]); */
}

costmax=cost[0];
imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]>costmax)
    {
        costmax=cost[i];
        imin=i;
    }
}

assignd(best,x1[imin]);
assignd(bestit,x1[imin]);

while (count<gen_max)
{
    count++;
    imin=0;

    for (i=0;i<NP;i++)
    {
        do a=int (rnd_uni(&rnd_uni_init)*NP); while (a==i);
        /*printf("a=%d ",a);*/

        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        /*printf("\nb=%d ",b);*/

        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        /*printf("\n c=%d",c); */
    }

    /*-----de/rand/1/bin-----*/

    j=int (rnd_uni(&rnd_uni_init)*D);
    /*printf(" j=%d",j);
    getch(); */

    for (k=1;k<=D;k++)
    {
        if ((rnd_uni(&rnd_uni_init)<CR || k==D)
        {
            trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
        }
        else trial[j]=(x1[i][j]+x1[i+1][j])/2.0;
    }
}

```

```

        if(trial[j]<0.0)   trial[j]=0.0;
        if(trial[j]>10.0) trial[j]=10.0;

        /*printf("r1=%f ,trial[%d]=%f,   ",r1,j,trial[j]);
           getch();*/
        j=(j+1)%D;
    }

    r1=trial[0];
    cost_trial=RKM(r1);   trial[0]=x;

/* printf("\ntrialcost=%f , cost[%d]=%f ",cost_trial,i,cost[i]);
   getch();*/

        if (cost_trial>=cost[i])
        {
            for (j=0;j<D;j++)
                x2[i][j]=trial[j];
            cost[i]=cost_trial;
            if(cost_trial>costmax)
            {
                costmax=cost_trial;
                imin=i;
                assignd(best,trial);
            }
        }
        else for (j=0;j<D;j++)

            x2[i][j]=x1[i][j];

/* printf("x1=%f   x2=%f   ",x2[i][0],x2[i][1]);
   getch(); */

} /*-----end of FOR loop after while-----*/

    assignd(bestit,best);
    for (i=0;i<NP;i++)
    {
        for (j=0;j<D;j++)
            x1[i][j]=x2[i][j];
    }
    costmax=cost[0];
    imin=0;
    for(i=1;i<NP;i++)
    {
        if(cost[i]>costmax)
        {
            costmax=cost[i];
            imin=i;
        }
    }

    costmin=cost[0];
    imin=0;
    for(i=1;i<NP;i++)
    {
        if(cost[i]<costmin)
        {
            costmin=cost[i];
            imin=i;
        }
    }

    if((costmax-costmin)<0.00001)

```

```

        break;

; /*-----end of while loop-----*/
end = clock();
printf("The time was: %f\n", (end - start) / CLK_TCK);
for(i=0;i<NP;i++)
    {
        printf("x1=%f ",x1[i][0]);
        printf("cost[%d]=%f ",i,cost[i]);
    }
printf("\ncmax=%f\n",costmax);
printf("\ncount=%d\n",count);
printf("bestx=%f",best[0]);
printf("\n NFE=%ld\n",nfe);

; /*-----end of main()-----*/

```

### Code of drying process problem using DE of Chapter-3

```

#define gen_max 1000
#define D 2
#define NP 20
#define F 0.7
#define CR 0.99
#define inibound_l 0
#define inibound_h 1

/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>

double evaluate(double [],long *);
double evaluate(double tmp[],long *nfe)
{ double cost,p1,p2,p3; (*nfe)++; /*** tmp[0]=x1, tmp[1]=x2;*****/

    p1=(1-exp(-107.9*tmp[1])/pow((tmp[0]*1000.0),0.41));
    p2=(9.27*pow((tmp[0]*1000.0),0.41)/(10000.0*tmp[1]));
    p3=(1-exp(-5.39*tmp[1])/pow((tmp[0]*1000.0),0.41));

    cost=(0.033*tmp[0]*1000.0)/((0.036/p1)+0.095+(p2*log(p1/p3)));
    return(cost);

} /****** end of evaluate() *****/

float rnd_uni(long *);
float rnd_uni(long *idum)
{
    long j; long k;

```

```

static long idum2=123456789;
static long iy=0;static long iv[NTAB]; float temp;
if(*idum<=0)
{
    if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
    for(j=NTAB+7;j>=0;j--)
    {
        k=(*idum)/IQ1;
        *idum=IA1*( *idum-k*IQ1)-k*IR1;
        if(*idum<0) *idum+=IM1;
        if(j<NTAB) iv[j]=*idum;
    }
    iy=iv[0];
}
k=(*idum)/IQ1;
*idum=IA1*( *idum-k*IQ1)-k*IR1;
if(*idum<0) *idum+=IM1;
k=idum2/IQ2;
idum2=IA2*(idum2-k*IQ2)-k*IR2;
if(idum2<0) idum2+=IM2;
j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
if(iy<1) iy+=IMM1;
if((temp=AM*iy)>RNMX) return RNMX;
else return temp;
}
void main()
{
    int i,j,k,a,b,c,good,count=0,seed; long nfe=0;
    double x1[NP][D],x2[NP][D],cost[NP],trial[D],cost_trial,costmax,costmin;
    clock_t start, end; double pen,lhs1,lhs2,lhs3,p1,p2,p3,p4;
    printf("\nseed=");
    scanf("%d",&seed);
    long rnd_uni_init= -(long)seed; start = clock();

    for (i=0;i<NP;i++)
    {
        for (j=0;j<D;j++)
        {
            x1[i][j]=inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h-inibound_l);
        }
    }
    pen=0.0;
    lhs1=0.2-(4.62*x1[i][1]*pow((x1[i][0]*1000),2.85)*pow(10,-10))-(1.055*x1[i][0]*1000.0/10000.0);
    if(lhs1<0.0)
    {
        pen=-lhs1*1e8;
        cost[i]=evaluate(x1[i], &nfe);
        cost[i]=cost[i]-pen;
        continue;
    }
    lhs2=(4.0/12.0)-(8.2*pow(10,-7)*pow((x1[i][0]*1000),1.85)*x1[i][1])-(2.25/12);
    if(lhs2<0.0)
    {
        pen=-lhs2*1e8;
        cost[i]=evaluate(x1[i], &nfe);
        cost[i]=cost[i]-pen;
        continue;
    }
    p1=(1-exp(-107.9*x1[i][1]/pow((x1[i][0]*1000.0),0.41)));
    p2=(9.27*pow((x1[i][0]*1000.0),0.41)/(10000.0*x1[i][1]));
    p3=(1-exp(-5.39*x1[i][1]/pow((x1[i][0]*1000.0),0.41)));
    p4=(109.6*x1[i][1]/pow((x1[i][0]*1000),0.41));
    lhs3=(2.0*0.32)-(p4*((0.036/p1)+0.095+(p2*log(p1/p3))));
    if(lhs3<0.0)
    {

```

```

pen=-lhs3*1e8;
cost[i]=evaluate(x1[i], &nfe);
cost[i]=cost[i]-pen;
continue;
}
if(lhs1>=0.0 && lhs2>=0.0 && lhs3>=0.0)
cost[i]=evaluate(x1[i], &nfe);
}
costmax=cost[0];
for(i=1;i<NP;i++)
{
    if(costmax<cost[i])
        costmax=cost[i];
}
while (count<gen_max)
{
    for (i=0;i<NP;i++)
    {
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if(rnd_uni(&rnd_uni_init)<CR || k==D)
            {
                trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
            }
            else trial[j]=x1[i][j];

            // if(trial[j]>1.0) trial[j]=1.0;
            // if(trial[j]<0.0) trial[j]=0.0000001;
            if(trial[j]<0.0 || trial[j]>1.0) trial[j]=0.0000001+rnd_uni(&rnd_uni_init);
            j=(j+1)%D;
        }
    }
pen=0.0;
lhs1=0.2-(4.62*trial[1]*pow((trial[0]*1000),2.85)*pow(10,-10))-(1.055*trial[0]*1000*pow(10,-4));

if(lhs1<0.0)
{
    pen=-lhs1*1e8;
    cost_trial=evaluate(trial, &nfe);
    cost_trial=cost_trial-pen;
    if(cost_trial>=cost[i])
    {
        for (j=0;j<D;j++)
            x2[i][j]=trial[j];
        cost[i]=cost_trial;
        if(cost_trial>costmax)
        {
            costmax=cost_trial;
            /* imin=i;
            assignd(best,trial); */
        }
    }
    else for (j=0;j<D;j++)
        x2[i][j]=x1[i][j];
    continue;
}
lhs2=(4.0/12.0)-(8.2*pow(10,-7)*pow((trial[0]*1000),1.85)*trial[1])-(2.25/12.0);
if(lhs2<0.0)
{
    pen=-lhs2*1e8;
    cost_trial=evaluate(trial, &nfe);
    cost_trial=cost_trial-pen;
    if(cost_trial>=cost[i])
    {

```

```

                for (j=0;j<D;j++)
                    x2[i][j]=trial[j];
                cost[i]=cost_trial;
                if(cost_trial>costmax)
                {
                    costmax=cost_trial;
                }
            }
            else for (j=0;j<D;j++)
                x2[i][j]=x1[i][j];
        continue;
    }
    p1=(1-exp(-107.9*trial[1]/pow((trial[0]*1000.0),0.41)));
    p2=(9.27*pow((trial[0]*1000.0),0.41)/(10000.0*trial[1]));
    p3=(1-exp(-5.39*trial[1]/pow((trial[0]*1000.0),0.41)));
    p4=(109.6*trial[1]/pow((trial[0]*1000.0),0.41));

    lhs3=(2.0*0.32)-(p4*((0.036/p1)+0.095+(p2*log(p1/p3))));
    if(lhs3<0.0)
    {
        pen=-lhs3*1e8;
        cost_trial=evaluate(trial, &nfe);
        cost_trial=cost_trial-pen;
        if(cost_trial>=cost[i])
        {
            for (j=0;j<D;j++)
                x2[i][j]=trial[j];
            cost[i]=cost_trial;
            if(cost_trial>costmax)
            {
                costmax=cost_trial;
                /* imin=i;
                assignd(best,trial); */
            }
        }
        else for (j=0;j<D;j++)
            x2[i][j]=x1[i][j];
        continue;
    }
    if(lhs1>=0.0 && lhs2>=0.0 && lhs3>=0.0)
        cost_trial=evaluate(trial, &nfe);

    if(cost_trial>=cost[i])
    {
        for (j=0;j<D;j++)
            x2[i][j]=trial[j];
        cost[i]=cost_trial;
        if(cost_trial>costmax)
        {
            costmax=cost_trial;
            /* imin=i;
            assignd(best,trial); */
        }
    }
    else for (j=0;j<D;j++)
        x2[i][j]=x1[i][j];
} /***** end of for loop *****/
for (i=0;i<NP;i++)
{
    for (j=0;j<D;j++)
        x1[i][j]=x2[i][j];
}
costmax=cost[0];
for(i=1;i<NP;i++)
{ if(costmax<cost[i])
    costmax=cost[i];
}

```

```

    }
    costmin=cost[0];
    for(i=1;i<NP;i++)
    { if(costmin>cost[i])
      costmin=cost[i];
    }

    if((costmax-costmin)<0.0001)
    break;
    count++;

} /***** end of while loop *****/

    end = clock();
    for(i=0;i<NP;i++)
    {
    for(j=0;j<D;j++)
    {if(j==0)
    printf("u[%d]=%lf" j,(x1[i][j]*1000));
    else
    printf("u[%d]=%lf" j,x1[i][j]);
    }
    printf("cost[%d]=%lf ",i,cost[i]);
    }
    printf("NFE=%ld\n",nfe);
    printf("The time was: %f\n", (end - start) / CLK_TCK);
    printf("costmax=%lf\n",costmax);
    printf("lhs1=%lf, lhs2=%lf lhs3=%lf\n",lhs1,lhs2,lhs3);
} /***** end of main() *****/

```

### Code of Heat Exchanger Network Design problem using DE (Chapter-3)

```

#define gen_max 2000
#define D 5
#define NP 50
#define F 0.5 /* 0.64 & 0.51*/
#define CR 0.8
#define inibound_l 10
#define inibound_h 10000
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#include<memory.h>
float evaluate(double [], double pen, long *);
float evaluate(double tmp[], double pen, long *nfe)
{ double cost;
  (*nfe)++;
  cost=tmp[0]+tmp[1]+tmp[2]+pen;
}

```

```

        return cost;
    }
void assignd(double a[], double b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}
float rnd_uni(long *);
float rnd_uni(long *idum)
{
    long j, long k;
    static long idum2=123456789;
    static long iy=0;static long iv[NTAB]; float temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;
            *idum=IA1*( *idum-k*IQ1)-k*IR1;
            if(*idum<0) *idum+=IM1;
            if(j<NTAB) iv[j]=*idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*( *idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
    j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
    if(iy<1) iy+=IMM1;
    if((temp=AM*iy)>RNMXX) return RNMXX;
    else return temp;
}
void main()
{
    int i,j,k,a,b,c,good,count=0,seed,imin; long nfe=0;
    double x1[NP][D],x2[NP][D],cost[NP],lhs4,lhs5,lhs6,trial[D];
    double cost_trial,cost4,pen,costmin,costmax,best[D],bestit[D];
    clock_t start, end;
    printf("\nseed=");
    scanf("%d",&seed);
    long rnd_uni_init= -(long)seed; start= clock();
    for (i=0;i<NP;i++)
    {
        for(j=0;j<D;j++)
        {
            x1[i][j]=inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h-inibound_l);
        }
        if(x1[i][0]<100.0 || x1[i][0]>10000.0) x1[i][0]=100.0+rnd_uni(&rnd_uni_init)*10000.0;
        if(x1[i][1]<1000.0 || x1[i][1]>10000.0) x1[i][1]=1000.0+rnd_uni(&rnd_uni_init)*10000.0;
        if(x1[i][2]<1000.0 || x1[i][2]>10000.0) x1[i][2]=1000.0+rnd_uni(&rnd_uni_init)*10000.0;
        for(j=3;j<D;j++)
        {
            if(x1[i][j]>1000.0 || x1[i][j]<10.0) x1[i][j]=10.0+rnd_uni(&rnd_uni_init)*1000.0;
        }
        /* if(x1[i][0]>10000.0) x1[i][0]=10000.0;
        if(x1[i][0]<100.0) x1[i][0]=100.0;

        if(x1[i][1]>10000.0) x1[i][1]=10000.0;
        if(x1[i][1]<1000.0) x1[i][1]=1000.0;

```



```

if(x1[i][2]>10000.0)          x1[i][2]=10000.0;
if(x1[i][2]<1000.0)         x1[i][2]=1000.0;

for(j=3;j<D;j++)
{
    if(x1[i][j]>1000.0)       x1[i][j]=1000.0;
    if(x1[i][j]<10.0)        x1[i][j]=10.0;
}
/*
pen=0.0;
lhs4=(100*x1[i][0])-(x1[i][0]*(400-x1[i][3]))+(833.33252*x1[i][3])-(83333.333);
if(lhs4>0.0)
{
    pen=(lhs4*1e7);
    cost[i]=evaluate(x1[i],pen,&nfe);
    continue;
}
lhs5=(x1[i][1]*x1[i][3])-(x1[i][1]*(400-x1[i][4]+x1[i][3]))-(1250*x1[i][3])+(1250*x1[i][4]);
if(lhs5>0.0)
{
    pen=(lhs5*1e7);
    cost[i]=evaluate(x1[i],pen,&nfe);
    continue;
}
lhs6=(x1[i][2]*x1[i][4])-(x1[i][2]*(100+x1[i][4]))-(2500*x1[i][4])+1250000.0;
if(lhs6>0.0)
{
    pen=(lhs6*1e7);
    cost[i]=evaluate(x1[i],pen,&nfe);
    continue;
}
if(lhs4<=0.0 && lhs5<=0.0 && lhs6<=0.0)
{
    cost[i]=evaluate(x1[i],pen,&nfe);
}
}
costmin=cost[0];
imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]<costmin)
    {
        costmin=cost[i];
        imin=i;
    }
}
assignd(best,x1[imin]);
assignd(bestit,x1[imin]);
while (count<gen_max)
{
    for (i=0;i<NP;i++)
    {
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        j=int (rnd_uni(&rnd_uni_init)*D); /*F=rnd_uni(&rnd_uni_init);*/
        for(k=1;k<=D;k++)
        {
            if(rnd_uni(&rnd_uni_init)<CR || k==D)
            {
                trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
            }
            else trial[j]=x1[i][j];
        }
        j=(j+1)%D;
    }
    if(trial[0]<100.0 || trial[0]>10000.0) trial[0]=100.0+rnd_uni(&rnd_uni_init)*10000.0;
    if(trial[1]<1000.0 || trial[1]>10000.0) trial[1]=1000.0+rnd_uni(&rnd_uni_init)*10000.0;
    if(trial[2]<1000.0 || trial[2]>10000.0) trial[2]=1000.0+rnd_uni(&rnd_uni_init)*10000.0;
    for(j=3;j<D;j++)
    {

```

```

        if(trial[j]>1000.0 || trial[j]<10.0)        trial[j]=10.0+rand_uni(&rand_uni_init)*1000.0;
        }
/* if(trial[0]>10000.0)        trial[0]=10000.0;
if(trial[0]<100.0)        trial[0]=100.0;
if(trial[1]>10000.0)        trial[1]=10000.0;
if(trial[1]<1000.0)        trial[1]=1000.0;
if(trial[2]>10000.0)        trial[2]=10000.0;
if(trial[2]<1000.0)        trial[2]=1000.0;

for(j=3;j<D;j++)
{
if(trial[j]>1000.0) trial[j]=1000.0;
if(trial[j]<10.0) trial[j]=10.0;
}*/
pen=0.0;
lhs4=(100*trial[0])-(trial[0]*(400.0-trial[3]))+(833.33252*trial[3])-(83333.333);
if(lhs4>0.0)
{
pen=(lhs4*1e7);
cost_trial=evaluate(trial,pen,&nfe);

if (cost_trial<=cost[i])
{
for (j=0;j<D;j++)
x2[i][j]=trial[j];
cost[i]=cost_trial;
if(cost_trial<costmin)
{
costmin=cost_trial;
imin=i;
assignd(best,trial);
}
}
else for (j=0;j<D;j++)
x2[i][j]=x1[i][j];

continue;
}
lhs5=(trial[1]*trial[3])-(trial[1]*(400.0+trial[3]-trial[4]))-(1250*trial[3])+(1250*trial[4]);
if(lhs5>0.0)
{
pen=(lhs5*1e7);
cost_trial=evaluate(trial,pen,&nfe);

if (cost_trial<=cost[i])
{
for (j=0;j<D;j++)
x2[i][j]=trial[j];
cost[i]=cost_trial;
if(cost_trial<costmin)
{
costmin=cost_trial;
imin=i;
assignd(best,trial);
}
}
else for (j=0;j<D;j++)
x2[i][j]=x1[i][j];

continue;
}
lhs6=(trial[2]*trial[4])-(trial[2]*(100+trial[4]))-(2500*trial[4])+1250000.0;
if(lhs6>0.0)
{
pen=(lhs6*1e7);
cost_trial=evaluate(trial,pen,&nfe);

if (cost_trial<=cost[i])
{
for (j=0;j<D;j++)
x2[i][j]=trial[j];

```

```

        cost[i]=cost_trial;
        if(cost_trial<costmin)
        {
            costmin=cost_trial;
            imin=i;
            assignd(best,trial);
        }
    }
    else for (j=0;j<D;j++)
        x2[i][j]=x1[i][j];
    continue;
}
if(lhs4<=0.0 && lhs5<=0.0 && lhs6<=0.0)
{
    cost_trial=evaluate(trial,pen,&nfe);
    /* printf("cost_trial=%f\n",cost_trial); */
    if (cost_trial<=cost[i])
    {
        for (j=0;j<D;j++)
            x2[i][j]=trial[j];
        cost[i]=cost_trial;
        if(cost_trial<costmin)
        {
            costmin=cost_trial;
            imin=i;
            assignd(best,trial);
        }
    }
    else for (j=0;j<D;j++)
        x2[i][j]=x1[i][j];
    continue;
}
}
/*****end of for loop *****/
assignd(bestit,best);
for(i=0;i<NP;i++)
{
    for (j=0;j<D;j++)
        x1[i][j]=x2[i][j];
}
costmin=cost[0];
for(i=1;i<NP;i++)
{
    if(costmin>cost[i])
        costmin=cost[i];
}
costmax=cost[0];
for(i=1;i<NP;i++)
{ if(costmax<cost[i])
    costmax=cost[i];
}
if((costmax-costmin)<0.00001)
    break;
count++;
}
/*****end of while loop*****/
end = clock();
for(i=0;i<NP;i++)
{
    printf("x1=%f x2=%f x3=%f x4=%f x5=%f",x1[i][0],x1[i][1],x1[i][2],x1[i][3],x1[i][4]);
    printf("cost[%d]=%f ",i,cost[i]);
}
printf("lhs4=%f lhs5=%f lhs6=%f \n ",lhs4,lhs5,lhs6);
printf("NFE=%ld\n",nfe);
printf("The time was: %f\n", (end - start) / CLK_TCK);
printf("costmin=%f costmax=%f\n",costmin,costmax);
}
/*****end of main() *****/

```

Code for Problem-1 of Chapter – 4 is given in this appendix.

### Code for Problem-1 of Chapter-4

```

#define gen_max 5000
#define D 2
#define NP 20
#define F 0.5
#define CR 0.8 /* 0.7 */
#define inibound_l 1.0
#define inibound_h 1.0
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
double evaluate(double [],long *);
double evaluate(double tmp[],long *nfe)
{ double cost,lhs1,lhs2,lhs3; (*nfe)++;
  lhs1=(1.25-(tmp[0]*tmp[0]*1.6*1.6)-(tmp[1]));
  lhs2=(tmp[0]*1.6+(tmp[1]));
  lhs3=tmp[1]*(tmp[1]-1.0);
  if(lhs1<=0.0) lhs1=0.0;
  if(lhs2<=1.6) lhs2=0.0;

  cost=(2*tmp[0]*1.6+(tmp[1]))+(fabs(lhs1)+lhs2+fabs(lhs3))*10;

  return(cost);
} /***** end of evaluate() *****/
float rnd_uni(long *);
float rnd_uni(long *idum)
{
  long j; long k;
  static long idum2=123456789;
  static long iy=0;static long iv[NTAB]; float temp;
  if(*idum<=0)
  {
    if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
    for(j=NTAB+7;j>=0;j--)
    {
      k=(*idum)/IQ1;
      *idum=IA1*( *idum-k*IQ1)-k*IR1;
      if(*idum<0) *idum+=IM1;
    }
  }
}

```

```

        if(j<NTAB)   iv[j]=*idum;
        }
        iy=iv[0];
    }
    k>(*idum)/IQ1;
    *idum=IA1>(*idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
    j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
    if(iy<1) iy+=IMM1;
    if((temp=AM*iy)>RNMXX) return RNMXX;
    else return temp;
}

void main()

{
    int i,j,k,a,b,c,good,count=0,seed; long nfe=0;
    double x1[NP][D],cost[NP],trial[D],cost_trial,costmax,costmin;
    clock_t start, end;
    printf("\nseed=");
    scanf("%d",&seed);
    long rnd_uni_init= -(long)seed; start = clock();

    for (i=0;i<NP;i++)
    {
        for (j=0;j<D;j++)
        {
            x1[i][j]=inibound_l + rnd_uni(&rnd_uni_init)*(inibound_h-inibound_l);
            // if(x1[i][j]>=0.5) x1[i][j]=1.0; else x1[i][j]=0.0;
        }
        cost[i]=evaluate(x1[i], &nfe);
    }
    costmin=cost[0];
    for(i=1;i<NP;i++)
    {
        if(costmin>cost[i])
            costmin=cost[i];
    }
    while (count<gen_max)
    {
        i=int (rnd_uni(&rnd_uni_init)*NP);
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if(rnd_uni(&rnd_uni_init)<CR || k==D)
            {
                trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
            }
            else trial[j]=x1[i][j];

            if(trial[0]<0.0 || trial[0]>1.6)
                trial[0]=rnd_uni(&rnd_uni_init);
            if(trial[1]<0.0 || trial[1]>1.0)
                trial[1]= rnd_uni(&rnd_uni_init);
            // if(trial[1]>=0.5) trial[1]=1.0; else trial[1]=0.0;

            j=(j+1)%D;
        }
        cost_trial=evaluate(trial, &nfe);
        if(cost_trial<cost[i])
        {

```

```

        for (j=0;j<D;j++)
            x1[i][j]=trial[j];
            cost[i]=cost_trial;
        if(cost_trial<costmin)
        {
            costmin=cost_trial;
            /* imin=i;
            assignd(best,trial); */
        }
    }
    costmax=cost[0];
    for(i=1;i<NP;i++)
    { if(costmax<cost[i])
      costmax=cost[i];
    }
    costmin=cost[0];
    for(i=1;i<NP;i++)
    { if(costmin>cost[i])
      costmin=cost[i];
    }
    count++;
    if((costmax-costmin)<0.00001)
        break;
} /***** end of while loop *****/
end = clock();
for(i=0;i<NP;i++)
{
    for(j=0;j<D;j++)
        if(j==0)
            printf("u[%d]=%lf",j,(x1[i][j]*1.6));
        else printf("u[%d]=%lf",j,(x1[i][j]));
        printf("cost[%d]=%lf ",i,cost[i]);
    }
    printf("NFE=%ld\n",nfe);
    printf("The time was: %f\n", (end - start) / CLK_TCK);
} /***** end of main() *****/

```

Code for test function ( $ES_2$ ) of Chapter – 5 is given in this appendix.

### MATLAB code for test function named $ES_2$ using HDE (Chapter – 5)

```

clear
stime=cputime;
rand('state',1);
global xqn
global nfe
xqn=[0 0];nfe=0.0;
D=2;
NP=D*10;
F=0.8;
CR=0.5;
TMAX=500;
OFopt=0.0;
lo=[-5 -5];
hi=[10 10];
t=0;
for i=1:NP
    for j=1:D
        x_de(i,j)=lo(j)+rand(1)*(hi(j)-lo(j));
        temp(j)=x_de(i,j);
    end
    cost(i)=prod(1(temp));nfe=nfe+1;
end
while (t<TMAX)
    for i=1:NP
        r1=i;
        while (r1==i)
            r1=int16(1+rand(1)*(NP));
            r1=double(r1);
        end
        r2=i;
        while ((r2==i)|(r2==r1))
            r2=int16(1+rand(1)*(NP));
            r2=double(r2);
        end
        r3=i;
        while ((r3==i)|(r3==r1)|(r3==r2))
            r3=int16(1+rand(1)*(NP));
            r3=double(r3);
        end
        j=int16(1+rand(1)*D);
        j=double(j);
        for k=1:D
            if ((rand(1)<CR)|(k==D))
                temp(j)=x_de(r3,j)+F*(x_de(r1,j)-x_de(r2,j));
            else temp(j)=x_de(i,j);
            end
            if ((temp(j)<lo(j))|(temp(j)>hi(j)))
                temp(j)=lo(j)+rand(1)*(hi(j)-lo(j));
            end
            j=mod(j,D)+1;
        end
        score=prod(1(temp));nfe=nfe+1;
        if (score<=cost(i))
            cost(i)=score;
    end
end

```

```

        for j=1:D
            x_de1(i,j)=temp(j);
        end
    else for j=1:D
        x_de1(i,j)=x_de(i,j);
    end
end
end % end of for loop after while
x_de=x_de1;

mini=1;
for i=2:NP
    if cost(i)<=cost(mini)
        mini=i;
    end
end
for j=1:D
    best(j)=x_de(mini,j);
end
x=best; bl=lo; bu=hi; ibound=0;
[x,f,bl,bu,ifail] = e04jaf(x,bl,bu,ibound);
    cost(mini)=f;
    xqn=x;
for j=1:D
    x_de(mini,j)=xqn(j);
end

    t=t+1;
    if (abs(OFopt-cost(mini))<abs(1e-4*OFopt+1e-6))
        break
    end

end % end of while loop
disp(['iteration number = ' int2str(t)])
disp(['x[1] = ' num2str(x_de(mini,1),12) ' ' x[2] = ' num2str(x_de(mini,2),10) ' ' cost = '
num2str(cost(mini),12)])
etime=cputime;
time=(etime-stime)
nfe
AAD=abs(cost(mini)-OFopt)

```



Codes for Multi-Product Batch Plant design of Chapter – 6 using NS-1 and NS-2 are given in this appendix.

## Code for MPBP problem using NS-1 (Chapter – 6)

```

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#define gen_max 10000
#define D 10
#define NP 100
#define F 0.4
#define CR 0.65
#define N 2
#define M 3
#define H 6000.0
#define alpha 250.0
#define beta 0.6
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
double evaluate(double [],long *);
double evaluate(double tmp[],long *nfe)
{ double cost,cost1,Q[2]; (*nfe)++;
double lhs1,lhs2,lhs3,lhs4,lhs5,lhs6,lhs7,lhs8,lhs9,lhs10,lhs11,lhs12,lhs13;
Q[0]=40000.0;Q[1]=20000.0;

tmp[0]=floor(tmp[0]); tmp[1]=floor(tmp[1]); tmp[2]=floor(tmp[2]);

lhs1=(Q[0]*(tmp[6]*20.0)/(H*(tmp[8]*625.0)))+(Q[1]*(tmp[7]*16.0)/(H*(tmp[9]*1250.0/3.0)))-1.0;
lhs2=((tmp[3]*2500)/(tmp[8]*625))-2.0;
lhs3=((tmp[4]*2500)/(tmp[8]*625))-3.0;
lhs4=((tmp[5]*2500)/(tmp[8]*625))-4.0;
lhs5=((tmp[3]*2500)/(tmp[9]*1250.0/3))-4.0;
lhs6=((tmp[4]*2500)/(tmp[9]*1250.0/3))-6.0;
lhs7=((tmp[5]*2500)/(tmp[9]*1250.0/3))-8.0;
lhs8=(tmp[0]*(tmp[6]*20))-8.0;
lhs9=(tmp[1]*tmp[6]*20)-20.0;
lhs10=(tmp[2]*(tmp[6]*20))-8.0;
lhs11=(tmp[0]*(tmp[7]*16))-16.0;
lhs12=(tmp[1]*(tmp[7]*16))-4.0;
lhs13=(tmp[2]*(tmp[7]*16))-4.0;
if(lhs1<=0.0) lhs1=0.0;

```

```

        if(lhs2>=0.0)    lhs2=0.0;
        if(lhs3>=0.0)    lhs3=0.0;
        if(lhs4>=0.0)    lhs4=0.0;
        if(lhs5>=0.0)    lhs5=0.0;
        if(lhs6>=0.0)    lhs6=0.0;
        if(lhs7>=0.0)    lhs7=0.0;
        if(lhs8>=0.0)    lhs8=0.0;
        if(lhs9>=0.0)    lhs9=0.0;
        if(lhs10>=0.0)   lhs10=0.0;
        if(lhs11>=0.0)   lhs11=0.0;
        if(lhs12>=0.0)   lhs12=0.0;
        if(lhs13>=0.0)   lhs13=0.0;

    cost1=(tmp[0])*pow((tmp[3]*2500.0), beta)+(tmp[1])*pow((tmp[4]*2500), beta)+(tmp[2])*pow((tmp[5]*2500),
beta));

    cost=cost1*alpha+(fabs(lhs1)+fabs(lhs2)+fabs(lhs3)+fabs(lhs4)+fabs(lhs5)+fabs(lhs6)+fabs(lhs7)+fabs(lhs8)+fabs
(lhs9)+fabs(lhs10)+fabs(lhs11)+fabs(lhs12)+fabs(lhs13))*1e5;
    return(cost);

} /***** end of evaluate() *****/

void assignd(double a[], double b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}

double rnd_uni(long *);
double rnd_uni(long *idum)
{
    long j; long k;
    static long idum2=123456789;
    static long iy=0;static long iv[NTAB]; double temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;
            *idum=IA1*(*idum-k*IQ1)-k*IR1;
            if(*idum<0) *idum+=IM1;
            if(j<NTAB) iv[j]=*idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*(*idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
    j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
    if(iy<1) iy+=IMM1;
    if((temp=AM*iy)>RNMXX) return RNMXX;
    else return temp;
}

void main()
{
    int i,j,k,a,b,c,seed,imin; long nfe=0,count=0;
    double x1[NP][D],x2[NP][D],cost[NP],trial[D],cost_trial;
    double costmin,costmax,best[D],bestit[D];

    clock_t start, end;
    printf("\nseed=");

```

```

scanf("%d",&seed);
long rnd_uni_init= -(long)seed; start = clock();

for (i=0;i<NP;i++)
{
    for (j=0;j<D;j++)
    {
        if(j==0 ||j==1 ||j==2)
            x1[i][j]= 1.0+rnd_uni(&rnd_uni_init)*(3.0-1.0);
        if(j==3 ||j==4 ||j==5)
            x1[i][j]=0.1 + rnd_uni(&rnd_uni_init)*(1.0-0.1);
        if(j==6 ||j==7)
            x1[i][j]=1.0/3 + rnd_uni(&rnd_uni_init)*(1.0-(1.0/3));
        if(j==8)
            x1[i][j]=-(400.0*x1[i][6]/(625.0*3)) + rnd_uni(&rnd_uni_init)*(1.0-(400.0*x1[i][6]/(625.0*3)));
        if(j==9)
            x1[i][j]=16.0*x1[i][7]/125.0 + rnd_uni(&rnd_uni_init)*(1.0-(16.0*x1[i][7]/125.0));
    }

    cost[i]=evaluate(x1[i], &nfe);
}

costmin=cost[0];
imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]<costmin)
    {
        costmin=cost[i];
        imin=i;
    }
}

assignd(best,x1[imin]);
assignd(bestit,x1[imin]);
while (count<gen_max)
{
    for(i=0;i<NP;i++)
    {
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if(rnd_uni(&rnd_uni_init)<CR || k==D)
                trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j]);
            else trial[j]=bestit[j]+F*(x1[a][j]-x1[b][j]);

            if(j==0 ||j==1 ||j==2)
            {
                if(trial[j]<1.0 || trial[j]>3.0)
                    trial[j]=1.0+rnd_uni(&rnd_uni_init)*(3.0-1.0);
            }
            if(j==3 ||j==4 ||j==5)
            {
                if(trial[j]<0.1 || trial[j]>1.0)
                    trial[j]=(0.1+rnd_uni(&rnd_uni_init)*(1.0-0.1));
            }
            if(j==6 ||j==7)
            {
                if(trial[j]<(1.0/3.0) || trial[j]>1.0)
                    trial[j]=(1.0/3)+rnd_uni(&rnd_uni_init)*(1.0-(1.0/3));
            }
            if(j==8)
            {
                if(trial[j]<-(400.0*trial[j-2]/(625.0*3))||trial[j]>1.0)

```

```

trial[j]=((400.0*trial[j-2]/(625.0*3))+rnd_uni(&rnd_uni_init)*(1.0-(400.0*trial[j-
2]/(625.0*3)))));
    }
    if(j==9)
    {
        if(trial[j]<(16.0*trial[j-2]/125.0) || trial[j]>1.0)
            trial[j]=((16.0*trial[j-2]/125.0)+rnd_uni(&rnd_uni_init)*(1.0-(16.0*trial[j-
2]/125.0)));
    }
    j=(j+1)%D;
}
cost_trial=evaluate(trial, &nfe);
if(cost_trial<=cost[i])
{
    for (j=0;j<D;j++)
        x2[i][j]=trial[j];
    cost[i]=cost_trial;
    if(cost_trial<costmin)
    {
        costmin=cost_trial;
        imin=i;
        assignd(best,trial);
    }
}
else for(j=0;j<D;j++)
    x2[i][j]=x1[i][j];
} /****** end of for loop *****/
assignd(bestit,best);
for(i=0;i<NP;i++)
for(j=0;j<D;j++)
    x1[i][j]=x2[i][j];
costmax=cost[0];
for(i=1;i<NP;i++)
{ if(costmax<cost[i])
    costmax=cost[i];
}
costmin=cost[0];
for(i=1;i<NP;i++)
{ if(costmin>cost[i])
    costmin=cost[i];
}
count++;
if((costmax-costmin)<0.00001)
    break;
} /****** end of while loop *****/
end = clock();
for(i=0;i<NP;i++)
{
    for(j=0;j<D;j++)
    { if(j==3 || j==4 || j==5)
        printf("u[%d]=%lf      "j,(x1[i][j]*2500));
        if(j==6)
            printf("u[%d]=%lf      "j,(x1[i][j]*20));
            if(j==7)
                printf("u[%d]=%lf      "j,(x1[i][j]*16));
                if(j==8)
                    printf("u[%d]=%lf      "j,(x1[i][j]*625));
                    if(j==9)
                        printf("u[%d]=%lf      "j,(x1[i][j]*1250/3.0));
                        if(j==0 || j==1 || j==2)
                            printf("u[%d]=%lf      "j,(x1[i][j]));
                            }
                            printf("cost[%d]=%lf      ",i,cost[i]);
                            }
}
printf("NFE=%ld\n",nfe);
printf("The time was: %f\n", (double)(end - start) / CLK_TCK);
printf("costmax is=%lf",costmax);

```

```

// printf("lhs1=%lf lhs2=%lf lhs3=%lf lhs4=%lf lhs5=%lf lhs6=%lf",lhs1,lhs2,lhs3,lhs4,lhs5,lhs6);
// printf("lhs7=%lf lhs8=%lf lhs9=%lf lhs10=%lf lhs11=%lf lhs12=%lf",lhs7,lhs8,lhs9,lhs10,lhs11,lhs12);
// printf("lhs13=%lf",lhs13);
printf("\ncostmin=%lf seed=%d\n",costmin,seed);
} /***** end of main() *****/

```

## Code for MPBP problem using NS-2 (Chapter – 6)

```

#include<stdlib.h>
#include<stdio.h>
#include<time.h>
#include<math.h>
#include<conio.h>
#define gen_max 10000
#define D 10
#define NP 100
#define F 0.1
#define CR 0.45
#define N 2
#define M 3
#define H 6000.0
#define alpha 250.0
#define beta 0.6
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
double evaluate(double [],long *);
double evaluate(double tmp[],long *nfe)
{ double cost,cost1,Q[2]; (*nfe)++;
double lhs1,lhs2,lhs3,lhs4,lhs5,lhs6,lhs7,lhs8,lhs9,lhs10,lhs11,lhs12,lhs13;
Q[0]=40000.0;Q[1]=20000.0;

tmp[0]=floor(tmp[0]); tmp[1]=floor(tmp[1]); tmp[2]=floor(tmp[2]);
lhs1=(Q[0]*(tmp[6]*20.0)/(H*(tmp[8]*625.0)))+(Q[1]*(tmp[7]*16.0)/(H*(tmp[9]*1250.0/3.0)))-1.0;
lhs2=((tmp[3]*2500)/(tmp[8]*625))-2.0;
lhs3=((tmp[4]*2500)/(tmp[8]*625))-3.0;
lhs4=((tmp[5]*2500)/(tmp[8]*625))-4.0;
lhs5=((tmp[3]*2500)/(tmp[9]*1250.0/3))-4.0;
lhs6=((tmp[4]*2500)/(tmp[9]*1250.0/3))-6.0;
lhs7=((tmp[5]*2500)/(tmp[9]*1250.0/3))-8.0;
lhs8=(tmp[0]*(tmp[6]*20))-8.0;
lhs9=(tmp[1]*tmp[6]*20)-20.0;
lhs10=(tmp[2]*(tmp[6]*20))-8.0;
lhs11=(tmp[0]*(tmp[7]*16))-16.0;
lhs12=(tmp[1]*(tmp[7]*16))-4.0;
lhs13=(tmp[2]*(tmp[7]*16))-4.0;
if(lhs1<=0.0) lhs1=0.0;
if(lhs2>=0.0) lhs2=0.0;
if(lhs3>=0.0) lhs3=0.0;
if(lhs4>=0.0) lhs4=0.0;
if(lhs5>=0.0) lhs5=0.0;
if(lhs6>=0.0) lhs6=0.0;
if(lhs7>=0.0) lhs7=0.0;

```

```

        if(lhs8>=0.0)    lhs8=0.0;
        if(lhs9>=0.0)    lhs9=0.0;
        if(lhs10>=0.0)   lhs10=0.0;
        if(lhs11>=0.0)   lhs11=0.0;
        if(lhs12>=0.0)   lhs12=0.0;
        if(lhs13>=0.0)   lhs13=0.0;

        cost1=(tmp[0])*pow((tmp[3]*2500.0), beta)+(tmp[1])*pow((tmp[4]*2500), beta)+(tmp[2])*pow((tmp[5]*2500),
beta));
        cost=cost1*alpha+(fabs(lhs1)+fabs(lhs2)+fabs(lhs3)+fabs(lhs4)+fabs(lhs5)+fabs(lhs6)+fabs(lhs7)+fabs(lhs8)+fabs
(lhs9)+fabs(lhs10)+fabs(lhs11)+fabs(lhs12)+fabs(lhs13))*1e5;
        return(cost);
    } /***** end of evaluate() *****/
void assignd(double a[], double b[])
{
    int j;
    for(j=0;j<D;j++)
    {
        a[j]=b[j];
    }
}

double rnd_uni(long *);
double rnd_uni(long *idum)
{
    long j; long k;
    static long idum2=123456789;
    static long iy=0;static long iv[NTAB]; double temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;
            *idum=IA1*(idum-k*IQ1)-k*IR1;
            if(*idum<0) *idum+=IM1;
            if(j<NTAB) iv[j]=*idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*(idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
    j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
    if(iy<1) iy+=IMM1;
    if((temp=AM*iy)>RNMX) return RNMX;
    else return temp;
}

void main()
{
    int i,j,k,a,b,c,d,e,seed,imin; long nfe=0,count=0;
    double x1[NP][D],x2[NP][D],cost[NP],trial[D],cost_trial;
    double costmin,costmax,best[D],bestit[D];
    clock_t start, end;
    printf("\nseed=");
    scanf("%d",&seed);
    long rnd_uni_init= -(long)seed; start = clock();
    for (i=0;i<NP;i++)
    {
        for (j=0;j<D;j++)
        {
            if(j==0 ||j==1 ||j==2)
                x1[i][j]= 1.0+rnd_uni(&rnd_uni_init)*(3.0-1.0);
            if(j==3 ||j==4 ||j==5)
                x1[i][j]=0.1 + rnd_uni(&rnd_uni_init)*(1.0-0.1);

```

```

        if(j==6 || j==7)
            x1[i][j]=1.0/3 + rnd_uni(&rnd_uni_init)*(1.0-(1.0/3));
        if(j==8)
            x1[i][j]=(400.0*x1[i][6]/(625.0*3)) + rnd_uni(&rnd_uni_init)*(1.0-(400.0*x1[i][6]/(625.0*3)));
        if(j==9)
            x1[i][j]=16.0*x1[i][7]/125.0 + rnd_uni(&rnd_uni_init)*(1.0-(16.0*x1[i][7]/125.0));
    }
    cost[i]=evaluate(x1[i], &nfe);
}
costmin=cost[0];
imin=0;
for(i=1;i<NP;i++)
{
    if(cost[i]<costmin)
    {
        costmin=cost[i];
        imin=i;
    }
}
assignd(best,x1[imin]);
assignd(bestit,x1[imin]);
while (count<gen_max)
{
    for(i=0;i<NP;i++)
    {
        do a=int ((rnd_uni(&rnd_uni_init))*NP); while (a==i);
        do b=int (rnd_uni(&rnd_uni_init)*NP); while (b==i || b==a);
        do c=int (rnd_uni(&rnd_uni_init)*NP); while (c==i || c==a || c==b);
        do d=int (rnd_uni(&rnd_uni_init)*NP); while (d==i || d==a || d==b || d==c);
        do e=int (rnd_uni(&rnd_uni_init)*NP); while (e==i || e==a || e==b || e==c || e==d);
        j=int (rnd_uni(&rnd_uni_init)*D);
        for (k=1;k<=D;k++)
        {
            if(rnd_uni(&rnd_uni_init)<CR || k==D)
                trial[j]=x1[c][j]+F*(x1[a][j]-x1[b][j])+(1-F)*(x1[d][j]-x1[e][j]);
            else trial[j]=bestit[j]+F*(x1[d][j]-x1[e][j]);
            if(j==0 || j==1 || j==2)
            {
                if(trial[j]<1.0 || trial[j]>3.0)
                    trial[j]=1.0+rnd_uni(&rnd_uni_init)*(3.0-1.0);
            }
            if(j==3 || j==4 || j==5)
            {
                if(trial[j]<0.1 || trial[j]>1.0)
                    trial[j]=(0.1+rnd_uni(&rnd_uni_init)*(1.0-0.1));
            }
            if(j==6 || j==7)
            {
                if(trial[j]<(1.0/3.0) || trial[j]>1.0)
                    trial[j]=(1.0/3)+rnd_uni(&rnd_uni_init)*(1.0-(1.0/3));
            }
            if(j==8)
            {
                if(trial[j]<(400.0*trial[j-2]/(625.0*3))||trial[j]>1.0)
                    trial[j]==((400.0*trial[j-2]/(625.0*3))+rnd_uni(&rnd_uni_init)*(1.0-(400.0*trial[j-
2]/(625.0*3))));
            }
            if(j==9)
            {
                if(trial[j]<(16.0*trial[j-2]/125.0) || trial[j]>1.0)
                    trial[j]==((16.0*trial[j-2]/125.0)+rnd_uni(&rnd_uni_init)*(1.0-(16.0*trial[j-
2]/125.0)));
            }
            j=(j+1)%D;
        }
    }
}

```

```

cost_trial=evaluate(trial, &nfe);
    if(cost_trial<=cost[i])
        {
            for (j=0;j<D;j++)
                x2[i][j]=trial[j];
            cost[i]=cost_trial;
            if(cost_trial<costmin)
                {
                    costmin=cost_trial;
                    imin=i;
                    assignd(best,trial);
                }
        }
    else for(j=0;j<D;j++)
        x2[i][j]=x1[i][j];
} /***** end of for loop *****/
assignd(bestit,best);
for(i=0;i<NP;i++)
    for(j=0;j<D;j++)
        x1[i][j]=x2[i][j];
costmax=cost[0];
for(i=1;i<NP;i++)
    { if(costmax<cost[i])
      costmax=cost[i];
    }
costmin=cost[0];
for(i=1;i<NP;i++)
    { if(costmin>cost[i])
      costmin=cost[i];
    }
count++;
if((costmax-costmin)<0.00001)
    break;
} /***** end of while loop *****/
end = clock();
for(i=0;i<NP;i++)
    {
        for(j=0;j<D;j++)
            { if(j==3 || j==4 || j==5)
              printf("u[%d]=%lf      ",j,(x1[i][j]*2500));
              if(j==6)
              printf("u[%d]=%lf      ",j,(x1[i][j]*20));
              if(j==7)
              printf("u[%d]=%lf      ",j,(x1[i][j]*16));
              if(j==8)
              printf("u[%d]=%lf      ",j,(x1[i][j]*625));
              if(j==9)
              printf("u[%d]=%lf      ",j,(x1[i][j]*1250/3.0));
              if(j==0 || j==1 || j==2)
              printf("u[%d]=%lf      ",j,(x1[i][j]));
            }
        printf("cost[%d]=%lf      ",i,cost[i]);
    }
printf("NFE=%ld\n",nfe);
printf("The time was: %f\n", (double)(end - start) / CLK_TCK);
printf("costmax is=%lf",costmax);
// printf("lhs1=%lf lhs2=%lf lhs3=%lf lhs4=%lf lhs5=%lf lhs6=%lf",lhs1,lhs2,lhs3,lhs4,lhs5,lhs6);
// printf("lhs7=%lf lhs8=%lf lhs9=%lf lhs10=%lf lhs11=%lf lhs12=%lf",lhs7,lhs8,lhs9,lhs10,lhs11,lhs12);
// printf("lhs13=%lf",lhs13);
printf("\ncostmin=%lf seed=%d\n",costmin,seed);
} /***** end of main() *****/

```



Codes for Schaffer's function of Chapter – 7 using NSDE and MNSDE are given in this appendix.

### Code of Schaffer's function using NSDE (Chapter – 7)

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<conio.h>
#define NP 400
#define D 1
#define F 0.2
#define CR 0.5
#define gen_max 200
#define UL 1000
#define LL -1000
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
double costf(double x[]);
double costg(double x[]);
double rnd_uni(long *);
double rnd_uni(long *idum)
{
    long j; long k;
    static long idum2=123456789;
    static long iy=0;static long iv[NTAB]; double temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;
            *idum=IA1*( *idum-k*IQ1)-k*IR1;
            if(*idum<0) *idum+=IM1;
            if(j<NTAB) iv[j]=*idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*( *idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
}

```

```

        j=iy/NDIV;  iy=iv[j]-idum2;  iv[j]=*idum;
        if(iy<1)  iy+=IMM1;
        if((temp=AM*iy)>RNMXX)  return RNMXX;
        else  return temp;
    }
void main()
{
    FILE *fp;
    fp=fopen("schaffer-NSDE.xls","a+");
    long gen=0;  int i,j,a,b,c,k=0,x=0,count=0,seed;
    double oldpop[NP][D],cost1[NP],cost2[NP],nondom[NP][D],selected[NP][D];
    double trial[D],costtrialf,costtrialg,costtargetf,costtargetg;
    printf("\nseed=");
    scanf("%d",&seed);
    long rnd_uni_init=-(long)seed;
    fprintf(fp,"NP = %d      F = %f  CR = %f  Max_gen = %d      seed = %d\n",NP,F,CR,gen_max,seed);
    for(i=0;i<NP;i++)
        for(j=0;j<D;j++)
            oldpop[i][j]=LL+(rnd_uni(&rnd_uni_init))*(UL-LL);
    for(gen=0;gen<gen_max;gen++)
    {
        for(i=0;i<NP;i++)
        {
            do a=(int)((rnd_uni(&rnd_uni_init))*NP); while(a==i);
            do b=(int)(rnd_uni(&rnd_uni_init)*NP); while(b==i || b==a);
            do c=(int)(rnd_uni(&rnd_uni_init)*NP); while(c==i || c==a || c==b);
            j=(int)rnd_uni(&rnd_uni_init)*D;
            for(k=1;k<=D;k++)
            {
                if(rnd_uni(&rnd_uni_init)<CR || k==D)
                    trial[j]=oldpop[c][j]+F*(oldpop[b][j]-oldpop[a][j]);
                else trial[j]=oldpop[i][j];

                if(trial[j]>UL || trial[j]<LL)
                    trial[j]=LL+rnd_uni(&rnd_uni_init)*(UL-LL);
                j=(j+1)%D;
            }
            costtrialf=costf(trial);
            costtrialg=costg(trial);
            costtargetf=costf(oldpop[i]);
            costtargetg=costg(oldpop[i]);
            if(costtrialf<costtargetf && costtrialg<costtargetg)
                for(j=0;j<D;j++)
                    nondom[i][j]=trial[j];
            else
            {
                for(j=0;j<D;j++)
                    nondom[i][j]=oldpop[i][j];
            }
        } // end of second for loop
        for(i=0;i<NP;i++)
        {
            for(j=0;j<D;j++)
                oldpop[i][j]=nondom[i][j];
        }
    } // end of first for loop
    for(i=0;i<NP;i++)
    { cost1[i]=costf(oldpop[i]);
    }
    for(i=0;i<NP;i++)
    { cost2[i]=costg(oldpop[i]);
    }
    for(i=0;i<NP;i++)
    {
        for(j=0;j<NP;j++)
        {
            if(i==j)

```

```

        {
            j=i+1;
        }

        if(cost1[j]<cost1[i]&&cost2[j]<cost2[i])
        {
            oldpop[i][0]=-1000000;
            break;
        }
    }
}

k=0;
for(j=0;j<NP;j++)
{
    if(oldpop[j][0]!=-1000000)
    {
        selected[k][0]=oldpop[j][0];
        k++; count++;
    }
}

for(i=0;i<k;i++)
{
    printf("final values of x[%d] %lf\n",k,selected[i][0]);
}

for(i=0;i<k;i++)
{
    fprintf(fp,"final values of x[%d] %lf %lf\n",k,selected[i][0],costf(selected[i]),costg(selected[i]));
}
} // End of Main()
double costf(double x[])
{
    double f;
    f=x[0]*x[0];
    return f;
}
double costg(double x[])
{
    double g;
    g=((x[0]-2.0)*(x[0]-2.0));
    return g;
}

```

### Code for Schaffer's function using MNSDE (Chapter – 7)

```

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<conio.h>
#define NP 100
#define D 1
#define F 0.5
#define CR 1.0
#define gen_max 200
#define UL 1000
#define LL -1000
/*----Constant for rnd_uni()-----*/
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774

```

```

#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS1 1.2e-7
#define RNMX (1.0-EPS1)
double costf(double x[]);
double costg(double x[]);
double rnd_uni(long *);
double rnd_uni(long *idum)
{
    long j; long k;
    static long idum2=123456789;
    static long iy=0; static long iv[NTAB]; double temp;
    if(*idum<=0)
    {
        if(-(*idum)<1) *idum=1; else *idum=-(*idum); idum2=(*idum);
        for(j=NTAB+7;j>=0;j--)
        {
            k=(*idum)/IQ1;
            *idum=IA1*( *idum-k*IQ1)-k*IR1;
            if(*idum<0) *idum+=IM1;
            if(j<NTAB) iv[j]=*idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*( *idum-k*IQ1)-k*IR1;
    if(*idum<0) *idum+=IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if(idum2<0) idum2+=IM2;
    j=iy/NDIV; iy=iv[j]-idum2; iv[j]=*idum;
    if(iy<1) iy+=IMM1;
    if((temp=AM*iy)>RNMX) return RNMX;
    else return temp;
}

void main()
{
    FILE *fp;
    fp=fopen("MNSDE-CR-schaffer.xls","a+");
    long gen=0; int i,j,a,b,c,k=0,x=0,count=0,seed;
    double oldpop[NP][D],cost1[NP],cost2[NP],selected[NP][D];
    double trial[D],costtrialf,costtrialg,costtargetg,costtargetf;

    printf("\nseed=");
    scanf("%d",&seed);
    long rnd_uni_init=-(long)seed;
    fprintf(fp,"NP = %d F = %f CR = %f Max_gen = %d seed = %d\n",NP,F,CR,gen_max,seed);
    for(i=0;i<NP;i++)
        for(j=0;j<D;j++)
            oldpop[i][j]=LL+(rnd_uni(&rnd_uni_init))*(UL-LL);
    for(gen=0;gen<gen_max;gen++) //First for loop
    {
        for(i=0;i<NP;i++) // Second for loop
        {
            do a=(int)((rnd_uni(&rnd_uni_init))*NP); while(a==i);
            do b=(int)(rnd_uni(&rnd_uni_init)*NP); while(b==i || b==a);
            do c=(int)(rnd_uni(&rnd_uni_init)*NP); while(c==i || c==a || c==b);
            j=(int)rnd_uni(&rnd_uni_init)*D;
            for(k=1;k<=D;k++)
            {
                if(rnd_uni(&rnd_uni_init)<CR || k==D)
                    trial[j]=oldpop[c][j]+F*(oldpop[b][j]-oldpop[a][j]);
                else trial[j]=oldpop[i][j];

                if(trial[j]>UL || trial[j]<LL)
                    trial[j]=LL+rnd_uni(&rnd_uni_init)*(UL-LL);
            }
        }
    }
}

```

```

        j=(j+1)%D;
    }
    costtrialf=costf(trial);
    costtrialg=costg(trial);
    costtargetf=costf(oldpop[i]);
    costtargetg=costg(oldpop[i]);

    if(costtrialf<costtargetf && costtrialg<costtargetg)
        for(j=0;j<D;j++)
            oldpop[i][j]=trial[j];
    } // end of second for loop
} // end of first for loop
for(i=0;i<NP;i++)
{ cost1[i]=costf(oldpop[i]);
}
for(i=0;i<NP;i++)
{ cost2[i]=costg(oldpop[i]);
}
for(i=0;i<NP;i++)
{
    for(j=0;j<NP;j++)
    { if (i==j)
        {
            j=i+1;
        }

        if(cost1[j]<cost1[i]&&cost2[j]<cost2[i])
        {
            oldpop[i][0]=-1000000;
            break;
        }
    }
}
k=0;
for(j=0;j<NP;j++)
{
    if(oldpop[j][0]!=-1000000)
    {
        selected[k][0]=oldpop[j][0];
        k++;    count++;
    }
}
for(i=0;i<k;i++)
{
    printf("final values of x[%d] %lf\n",k,selected[i][0]);
    //fprintf(fp,"final values of x[%d] %lf\n",k,selected[i][0]);
}
for(i=0;i<k;i++)
    fprintf(fp,"final values of x[%d] %lf %lf\n",k,selected[i][0],costf(selected[i]),costg(selected[i]));
} // End of Main()
double costf(double x[])
{
    double f;
    f=x[0]*x[0];
    return f;
}
double costg(double x[])
{
    double g;
    g=((x[0]-2.0)*(x[0]-2.0));
    return g;
}

```