Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Electrical Engineering Department
Computer Science Section

On the Use of Self-Adaptation and Elitism for Multiobjective Particle Swarm Optimization

By:

Gregorio Toscano Pulido

as the fulfilment of the
requirement for the degree of:
Doctor of Science

Specialization in:
Electrical Engineering

Option:
Computer Science

Advisor:
Dr. Carlos A. Coello Coello

México City, México. September 29, 2005

Centro de Investigación y Estudios Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Ingeniería Eléctrica
Sección de Computación

Uso de Auto-Adaptación y Elitismo para Optimización Multiobjetivo Mediante Cúmulos de Partículas

Tesis que presenta:

Gregorio Toscano Pulido

para obtener el grado de:
Doctor en Ciencias

En especialidad de:
Ingeniería Eléctrica

Opción:
Computación

Director de tesis:
Dr. Carlos A. Coello Coello

México D.F., México. Septiembre 29 de 2005.

A mamá y papá

Acknowledgments

- I express my sincere gratitude to Dr. Carlos A. Coello Coello for his invaluable help and guidance over the last 5 years. Thanks for being my friend and mentor.
- I thank the reviewers for their valuable comments that greatly helped me to improve the contents of this dissertation.
- I thank my parents for their love from the first day of my life, for their support, financial and otherwise which have allowed me to enjoy student life for so long; my sisters Pao and Mari, and my cousin gabo to be always my friends and accomplices; and my beloved Gaby to be as she is.
- I acknowledge to all my friends at CINVESTAV: Adric, Agusto, Alex, Alfredo, Carlos, Efrén, Gustavo, Isaí, Jaimico, Ricardo, Luis, Lucio, Margarita, Nareli, Oscar, Francisco, Rolando and Israel. Thanks also to the rest of my friends (too many to name).
- I would like to thank to Sofy, Felipa and Flor for their administrative support.
- I acknowledge support from CONACyT through a scholarship to pursue graduate studies at the Computer Science Section of the Electrical Engineering Department at CINVESTAV-IPN, in México.
- The research work done in this PhD thesis was derived from CONACyT's project "Técnicas Avanzadas de Optimización Evolutiva Multiobjetivo" (Ref. 45683-Y) whose Principal Investigator is Dr. Carlos A. Coello Coello.

Abstract

Most real world problems are multiobjective. Usually, traditional nonlinear multiobjective optimization techniques are computationally expensive. Consequently, it is difficult to obtain solutions in polynomial time if we increase the complexity of the problem. Furthermore, traditional mathematical programming techniques are normally highly susceptible to the shape or continuity of the Pareto front. Therefore, alternative ways of thinking are needed, new algorithms – evolutionary computation.

Particle swarm optimization (PSO) is a relatively recent evolutionary optimization heuristic that has been found to be very successful in a wide variety of optimization tasks. Its high speed of convergence and its relative simplicity make PSO a highly viable candidate to be used for solving not only problems with a single objective function, but also multiobjective optimization problems. However, PSO lacks an explicit mechanism to manage multiple objectives.

In this dissertation, we analyze and extend the PSO algorithm to solve "efficiently" multiobjective optimization problems. Our research is divided into three main components: 1) a proposal which extends PSO to handle multiple objectives. The main novelty of the approach consists on using a clustering technique in order to divide the population of particles into several subswarms in variable space. Such modification, significantly improves the quality of the Pareto fronts produced, since in each subswarm emerge a local search behavior. Also, in order to reduce the non-dominated set, we propose an additional approach to decide whether a solution is accepted or not. 2) We present a mechanism to handle constraints with PSO. Our proposal uses a simple criterion based on closeness of a particle to the feasible region in order to select a leader. Our comparison of results indicates that the proposed approach is highly competitive with respect to three constraint-handling techniques representative of the state-of-the-art in the area. This constraint-handling approach was implemented into our multiobjective particle swarm optimization algorithm (MOPSO). 3) Finally, in order to improve the general performance of the algorithm, we performed an study about the MOPSO's parameters. Then, we proposed a self-adaptation scheme to select the best parameters' values; such proposal was validated using several test functions and metrics taken from the standard literature on evolutionary multiobjective optimization. The results indicate that our approach is a viable alternative since it outperformed some of

the best multiobjective evolutionary algorithms known to date.

Resumen

La mayoría de los problemas de mundo real son multiobjetivo. Generalmente las técnicas tradicionales de optimización no-lineal multiobjetivo son computacionalmente costosas. Consecuentemente, es difícil obtener soluciones en tiempo polinomial, si se incrementa la complejidad del problema. Además, las técnicas tradicionales de programación matemática son normalmente altamente susceptibles a la forma o continuidad del frente de Pareto. Por lo tanto, se necesitan nuevas herramientas alternativas para abordar dichos problemas. Una de ellas es la computación evolutiva.

La "optimización mediante cúmulos de partículas" (PSO[1]) es una heurística evolutiva relativamente nueva que ha sido usada para resolver exitosamente una amplia variedad de tareas de optimización. Su alta velocidad de convergencia y su relativa simplicidad hacen al PSO un candidato altamente viable para utilizarlo en la resolución de problemas multiobjetivo. Sin embargo, el PSO carece de un mecanismo explícito para manejar objetivos múltiples.

En esta tesis, analizamos y extendemos el PSO para resolver "eficientemente" problemas de optimización multiobjetivo. Nuestra investigación está dividida en tres partes principales: 1) una propuesta para habilitar al PSO para que pueda manejar objetivos múltiples. La principal novedad del enfoque consiste en el uso de una técnica de clustering para dividir la población en diferentes sub-cúmulos (en el espacio de las variables de decisión). Dicha modificación mejora significativamente la calidad del frente de Pareto producido, en parte, debido a que hace emerger una búsqueda local en cada sub-cúmulo. También, para reducir el conjunto de soluciones no-dominadas, proponemos un enfoque adicional para aceptar soluciones. 2) Presentamos un mecanismo para manejar restricciones usando el PSO. Nuestra propuesta usa un criterio de selección de líderes basado en la cercanía de una partícula a la región factible. Los resultados nos indican que el algoritmo propuesto es altamente competitivo con respecto a tres técnicas de manejo de restricciones representativas del estado del arte en el área. Nuestra técnica fue también implentada en nuestro PSO multiobjetivo (MOPSO). 3) Finalmente, para mejorar el desempeño general del algoritmo, realizamos un estudio sobre los parámetros del MOPSO. Proponemos un esquema de auto-adaptación para seleccionar los mejores valores de los parámetros; nuestra propuesta fue validada usando

[1]Por sus siglas en inglés.

varias funciones de prueba y métricas tomadas de la literatura especializada. Los resultados indican que nuestro algoritmo final es una alternativa viable, ya que supera el desempeño algunos de los mejores algoritmos multiobjetivos que se conocen a la fecha.

Contents

Acknowledgments **ii**

Abstract **iv**

Resumen **vii**

Table of Contents **ix**

1 Introduction **1**
1.1 Introduction . . . 1
1.2 Motivation . . . 1
1.3 Objectives . . . 2
1.4 Contributions . . . 2
1.5 Document Outline . . . 2

2 Background **5**
2.1 Introduction . . . 5
2.2 Optimization . . . 5
2.2.1 Global Optimization . . . 5
2.2.2 Multiobjective Optimization . . . 6
2.3 Background Concepts . . . 6
2.4 Performance Measures . . . 10
2.5 Test Functions . . . 13
2.5.1 Deb 1's Test Function . . . 13
2.5.2 Deb 2's Test Function . . . 13
2.5.3 Kursawe's Test Function . . . 14
2.5.4 DTLZ2's Test Function . . . 15
2.5.5 ZDT1's Test Function . . . 16
2.5.6 ZDT2's Test Function . . . 16
2.5.7 ZDT3's Test Function . . . 17

2.5.8 ZDT4's Test Function . . . 18
2.5.9 ZDT6's Test Function . . . 18
2.5.10 Kita's Test Function . . . 19
2.5.11 Welded Beam's Test Function . . . 20
2.5.12 Speed Reducer's Test Function . . . 23
2.5.13 Osyczka2 's Test Function . . . 25

3 Evolutionary Algorithms 27
3.1 Introduction . . . 27
3.2 Evolutionary Algorithms . . . 27
3.2.1 Evolutionary Programming . . . 29
3.2.2 Evolution Strategies . . . 29
3.2.3 Genetic Algorithms . . . 30
3.2.4 Genetic Programming . . . 31
3.2.5 Swarm Intelligence . . . 31
3.2.5.1 Particle Swarm Optimization . . . 31
3.3 Multiobjective Evolutionary Algorithms . . . 33
3.3.1 Multi-Objective Evolutionary Components . . . 33
3.3.2 Initialization . . . 34
3.3.3 Parent Selection Mechanism . . . 34
3.3.4 Variation Operators . . . 37
3.3.4.1 Mutation . . . 37
3.3.4.2 Recombination . . . 38
3.3.5 Survivor Selection Mechanism . . . 38
3.3.5.1 Using a Historical Archive of Solutions . . . 39

4 Multiobjective Optimization Techniques 41
4.1 Introduction . . . 41
4.2 Traditional Techniques . . . 41
4.2.1 No Preference Information . . . 41
4.2.2 *A priori* Methods . . . 42
4.2.3 *A posteriori* Methods . . . 42
4.2.4 Interactive Methods . . . 43
4.2.5 Drawbacks of the Conventional Techniques . . . 44
4.3 Evolutionary Techniques . . . 44
4.3.1 Vector Evaluated Genetic Algorithm (VEGA) . . . 45
4.3.2 Multiobjective Genetic Algorithm (MOGA) . . . 46
4.3.3 Niched Pareto Genetic Algorithm (NPGA) . . . 46
4.3.4 Non-dominated Sorting Genetic Algorithm (NSGA) . . . 47
4.3.5 Strength Pareto Evolutionary Algorithm (SPEA) . . . 48
4.3.6 Pareto Archived Evolution Strategy (PAES) . . . 50
4.3.7 Multiobjective Micro Genetic Algorithm (MicroGA) . . . 50
4.3.8 ϵ-MOEA . . . 51
4.3.9 Multiobjective Particle Swarm Optimization . . . 52
4.3.10 Current Trends . . . 57

4.3.11 Advantages and Disadvantages of Evolutionary Algorithms for MOPs 57

5 Multiobjective Particle Swarm Optimization 59
5.1 Multiobjective Particle Swarm Optimization 60
5.2 Handling multiple objectives 60
5.3 Improving the distribution 64
5.3.1 Adaptive Grid 64
5.3.2 ϵ-dominance 67
5.3.3 Hyper-plane distribution 68
5.3.4 Experiment 1 70
5.3.5 Experiment 2 75
5.4 Maximizing the Spread 78
5.4.1 Using Subswarms to Improve the Spread 80
5.5 MOPSO's Comparison of Results 84
5.5.1 Kursawe's Test Function 84
5.5.2 ZDT1' Test Function 86
5.5.3 ZDT2's Test Function 88
5.6 Conclusions 90

6 A Constraint-Handling Mechanism for PSO 91
6.1 Introduction 91
6.2 Related Work 92
6.3 Constrained Particle Swarm Optimization 92
6.3.1 Mechanism to Handle Constraints 93
6.4 Test Functions 94
6.5 Comparison of Results 98
6.6 A Constraint-Handling Mechanism for MOPSO 102
6.7 CMOPSO's Comparison of Results 105
6.7.1 Kita's Test Function 105
6.7.2 Speed Reducer's Test Function 107
6.7.3 Osyczka 2's Test Function 109
6.7.4 Welded Beam's Test Function 111
6.8 Conclusions 113

7 Parameter Control in Multiobjective Particle Swarm Optimization 115
7.1 Introduction 115
7.2 Tuning or Adapting Parameters 116
7.3 Parameter Adaptation 116
7.4 Related Work 117
7.5 MOPSO: Parameter's Analysis 118
7.5.1 Experiment 1 119
7.5.2 Experiment 2 119
7.5.3 Experiment 3 120
7.5.4 Conclusions from the Experiments 120
7.6 Self-Adaptation Mechanism 121

7.6.1 Deb1's Test Function . . . 123
7.6.2 Deb2's Test Function . . . 125
7.6.3 Kursawe's Test Function . . . 126
7.6.4 Kita's Test Function . . . 128
7.6.5 ZDT1's Test Function . . . 130
7.6.6 ZDT2's Test Function . . . 132
7.6.7 ZDT3's Test Function . . . 134
7.6.8 ZDT6's Test Function . . . 136
7.6.9 Welded Beam Test Function . . . 138
7.6.10 Osyczka2's Test Function . . . 140
7.6.11 Speed Reducer Test Function . . . 142
7.6.12 Conclusions from the Experiment . . . 144
7.7 Comparison of Results . . . 146
7.7.1 Kursawe's Test Function . . . 146
7.7.2 ZDT1's Test Function . . . 148
7.7.3 ZDT2's Test Function . . . 150
7.7.4 DTLZ2's Test Function . . . 152
7.7.5 Welded Beam Test Function . . . 154
7.7.6 Kita's Test Function . . . 156
7.7.7 Speed Reducer Test Function . . . 158
7.7.8 Osyczka2's Test Function . . . 160
7.8 Conclusions . . . 160

8 Final Remarks 163
8.1 Conclusions . . . 163
8.2 Future Work . . . 164

A 165
A.1 ZDT1's test function . . . 165
A.2 ZDT2's test function . . . 165
A.3 ZDT3's test function . . . 165
A.4 ZDT6's test function . . . 170
A.5 Kursawe's test function . . . 170
A.6 Deb's test function . . . 170
A.7 Deb2's test function . . . 176
A.8 Scheme 4 . . . 176

B 181
B.1 Experiment 1 . . . 181
B.2 Experiment 2 . . . 193
B.3 Experiment 3 . . . 205

C 213

Bibliografy 247

List of Figures

2.1 Example that is meant to show the difference between decision variable space (left) and objective function space (right). 8

2.2 Example that is meant to show the difference between the Pareto optimal set (left) and the Pareto front (right). 9

2.3 Example that is meant to show the difference between a false and a true Pareto front 10

2.4 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Deb 1's test function. . 14

2.5 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Deb 2's test function. . 14

2.6 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Kursawe's test function. 15

2.7 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for DTLZ2's test function. 16

2.8 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT1's test function. . 16

2.9 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT2's test function. . 17

2.10 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT3's test function. . 18

2.11 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT4's test function. . 18

2.12 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT6's test function. . 19

2.14 Welded Beam 20

2.13 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT6's test function. . 20

2.15 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Welded Beam's test function. 22
2.16 Plane truss used for the fourth example. The structural volume and the joint displacement (Δ) are to be minimized. 23
2.17 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Speed Reducer's test function. 24
2.18 The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Osyczka 2's test function. 26

3.1 The top figure shows the location of 4 swarms clustered in the variable space of Kursawe's test function (see Section 2.5.3), and the figure at the bottom shows the same in the objective function space. 35
3.2 Possible decision cases to store for the external archive. 40

4.1 A classification of some methods used to conduct multiobjective optimization using mathematical programming techniques 42
4.2 Graphical representation of the layers adopted by the NSGA. 48
4.3 Illustration of the ϵ-MOEA procedure 53

5.1 Graphical representation of the insertion of a new element in the adaptive grid when the individual lies within the current boundaries of the grid. . . . 64
5.2 Graphical representation of the insertion of a new element in the adaptive grid when this lies outside the previous boundaries of the grid. 65
5.3 Graphical representation of the adaptive grid. In this case, we used two objective functions for ease of understanding (assuming minimization). . . . 66
5.4 Graphical representation of the hyper-plane distribution. 69
5.5 Pareto fronts produced by a) adaptive grid, b) ϵ-Pareto, and c) hyper-plane distribution, for the ZDT1's test function. 72
5.6 Pareto fronts produced by a) adaptive grid, b) ϵ-Pareto, and c) hyper-plane distribution, for the ZDT2's test function. 73
5.7 Pareto fronts produced by a) adaptive grid, b) ϵ-Pareto, and c) hyper-plane distribution, for the ZDT3's test function. 74
5.8 Graphical representation of the insertion of a new element in the adaptive grid when the individual lies within the current boundaries of the grid. . . . 75
5.9 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Kursawe's test function. 85
5.10 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for the ZDT1's test function. 86
5.11 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for ZDT2's test function. 88

6.1 Graphical representation of our example that explains the constraint-handling mechanism incorporated into our PSO algorithm. 93

6.2 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for Kita's test function. 106
6.3 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for the Speed Reducer test function. 107
6.4 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for Osyczka 2's test function. 109
6.5 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for the Welded Beam test function. 111

7.1 Roulette Wheel Selection example at the a) first, b) second and c) third generation. 122
7.2 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for Kursawe's test function. 147
7.3 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for ZDT1's test function. 148
7.4 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for ZDT2's test function. 150
7.5 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for DTLZ2's test function. 152
7.6 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for the Welded Beam test function. 154
7.7 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Kita's test function. 156
7.8 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Speed reducer's test function. 158
7.9 Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Osyczka2's test function. 160

A.1 Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZDT1's test function. 166
A.2 Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles for ZDT1's test function. 167
A.3 Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZTD2's test function. 168
A.4 Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particles' position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for ZDT2's test function. 169
A.5 Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZTD3's test function. 170

A.6 Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for ZDT3's test function. 171
A.7 Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZTD6's test function. 172
A.8 Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for ZDT6's test function. 173
A.9 Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for Kursawe's test function. 174
A.10 Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for Kursawe's test function. 175
A.11 Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for Deb's test function. 176
A.12 Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for Deb's test function. 177
A.13 Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for Deb2's test function. 178
A.14 Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for DEB2's test function. 179
A.15 Pareto fronts produced by the scheme 4 for test functions 180

B.1 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kursawe's test function. 182
B.2 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb1's test function. 183
B.3 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb2's test function. 184
B.4 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT1's test function. 185
B.5 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT2's test function. 186

B.6 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT3's test function. 187
B.7 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT6's test function. 188
B.8 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kita's test function. 189
B.9 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Welded beam test function. 190
B.10 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Osyczka2's test function. 191
B.11 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Speed Reducer test function. 192
B.12 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kursawe's test function. 194
B.13 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb1's test function. 195
B.14 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb2's test function. 196
B.15 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT1's test function. 197
B.16 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT2's test function. 198
B.17 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT3's test function. 199
B.18 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT6's test function. 200
B.19 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kita's test function. 201
B.20 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Welded Beam test function. 202
B.21 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Osyczka2's test function. 203
B.22 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the the Speed Reducer test function. . 204
B.23 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kursawe's test function. 206
B.24 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb2's test function. 207
B.25 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT3's test function. 208
B.26 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kita's test function. 209
B.27 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Welded Beam test function. 210

B.28 Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Osyczka2's test function. 211

C.1 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb1's test function. 214

C.2 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb1's test function. 215

C.3 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb1's test function. 216

C.4 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb2's test function. 217

C.5 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb2's test function. 218

C.6 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb2's test function. 219

C.7 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kursawe's test function. 220

C.8 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a fixed , b random, c self-adaptation and d self-adaptation (using half of the parameters' range) mechanisms for Kursawe's test function. 221

C.9 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kursawe's test function. 222

C.10 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kita's test function. 223

C.11 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kita's test function. 224
C.12 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kita's test function. 225
C.13 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT1's test function. 226
C.14 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT1's test function. 227
C.15 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT1's test function. 228
C.16 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT2's test function. 229
C.17 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT2's test function. 230
C.18 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT2's test function. 231
C.19 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT3's test function. 232
C.20 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT3's test function. 233
C.21 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT3's test function. 234

C.22 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT6's test function. 235
C.23 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT6's test function. 236
C.24 Solutions produced using 250 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT6's test function. 237
C.25 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Welded Beam test function. 238
C.26 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Welded Beam test function. 239
C.27 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Welded Beam test function. 240
C.28 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Osyczka 2's test function. 241
C.29 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Osyczka 2's test function. 242
C.30 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Osyczka 2's test function. 243
C.31 Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Speed Reducer test function. 244
C.32 Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Speed Reducer test function. 245

C.33 Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Speed Reducer test function. 246

List of Tables

5.1 Comparison of results of the approaches used to provide a good distribution of solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to the Hyper Volume metric. 71
5.2 Comparison of results of the approaches used to proved a good distribution of solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to Spacing metric. 71
5.3 Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to the Inverted Generational Distance metric. 77
5.4 Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to the Inverted Generational Distance metric. 77
5.5 Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to Inverted Generational Distance metric. 79
5.6 Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to Inverted Generational Distance metric. 82
5.7 Statical results obtained from the counting where the particle was equal to its gbest. 83
5.8 Comparison of results of MOPSO with respect to Inverted Generational Distance metric. 83
5.9 Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for the Kursawe's test function. 84
5.10 Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Kursawe's test function. 84
5.11 Comparison of results of the MOPSO, NSGA-II and our ϵ-MOEA with respect to the Inverted Generational Distance (IGD) and Success Counting (SC) for Kursawe's test function. 85

5.12 Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT1's test function. 86
5.13 Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT1's test function. 87
5.14 Comparison of results of the MOPSO, NSGA-II and our ϵ-MOEA with respect to Inverted Generational Distance (IGD) and Success Counting (SC) for ZDT1's test function. 87
5.15 Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT2's test function. 88
5.16 Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT2's test function. 89
5.17 Comparison of results of the MOPSO, NSGA-II and our ϵ-MOEA with respect to the Inverted Generational Distance (IGD) and Success counting (SC) for ZDT2's test function. 89

6.1 Values of ρ for the 13 test problems chosen. 98
6.2 Comparison of our PSO algorithm with respect to the Homomorphous Maps (HM) [76]. 99
6.3 Comparison of results of our PSO with respect to Stochastic Ranking (SR) [104]. 100
6.4 Comparison of results of our PSO with respect to the Adaptive Segregational Constraint Handling Evolutionary Algorithm (ASCHEA) [48]. 101
6.5 Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for Kita's test function. 105
6.6 Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for Kita's test function. 105
6.7 Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success Counting(SC) for Kita's test function. 106
6.8 Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for the Speed Reducer test function. 107
6.9 Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for the Speed Reducer test function. 108
6.10 Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success counting(SC) for the Speed Reducer test function. 108
6.11 Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for Osyczka 2's test function. 109
6.12 Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for Osyczka 2's test function. 110
6.13 Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success counting(SC) for Osyczka 2's test function. 110
6.14 Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for the Welded Beam test function. 111

6.15 Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for the Welded Beam test function. 112

6.16 Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success counting(SC) for the Welded Beam test function. 112

7.1 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Deb1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 123

7.2 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Deb1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 124

7.3 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Deb2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 125

7.4 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Kursawe's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. . . 126

7.5 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Kursawe's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 127

7.6 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Kita's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 128

7.7 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Kita's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 129

7.8 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 130

7.9 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. . . 131

7.10 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 132
7.11 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. . . 133
7.12 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT3's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 134
7.13 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT3's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. . . 135
7.14 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT6's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 136
7.15 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT6's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. . . 137
7.16 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for the Welded Beam test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 138
7.17 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for the Welded Beam test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. . . 139
7.18 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Osyczka2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. . . 140
7.19 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Osyczka2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 141
7.20 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for the Speed Reducer test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 142

7.21 Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for the Speed Reducer test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed. 143
7.22 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Kursawe's test function. 146
7.23 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for Kursawe's test function. 147
7.24 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for Kursawe's test function. 147
7.25 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT1's test function. 148
7.26 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT1's test function. 149
7.27 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for ZDT1's test function. 149
7.28 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT2's test function. 150
7.29 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT2's test function. 151
7.30 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for ZDT2's test function. 151
7.31 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for DTLZ2's test function. 152
7.32 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for DTLZ2's test function. 153
7.33 Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for DTLZ2's test function. 153
7.34 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for the Welded Beam test function. 154
7.35 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for the Welded Beam test function. 155
7.36 Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for the Welded Beam test function. 155

7.37 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Kita's test function. 156
7.38 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for Kita's test function. 157
7.39 Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for Kita's test function. 157
7.40 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for the Speed Reducer test function. . . . 158
7.41 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for the Speed Reducer test function. 159
7.42 Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for the Speed Reducer test function. 159
7.43 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Osyczka2's test function. 160
7.44 Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for Osyczka2's test function. 161
7.45 Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for Osyczka2's test function. 161

1

Introduction

1.1 Introduction

MULTIOBJECTIVE optimization problems (e.g., those having two or more objectives to be optimized simultaneously) arise in all areas. Operational researchers have developed a variety of approaches to deal with multiobjective optimization problems. However, such approaches normally rely on specific features of the problem (e.g., some require that the objective function and the constraints are continuous), and have several limitations (i.e., most of them generate one solution per run). This motivates the use of alternative techniques.

Evolutionary computing comprises a set of algorithms which draw inspiration from the process of natural evolution. These algorithms present a series of suitable characteristics to solve multiobjective optimization problems. Currently, there is an specific research area within evolutionary computation that deals with multiobjective optimization problems: Evolutionary Multiobjective Optimization (EMO for short) which has significantly grown in the last few years, giving rise to a wide variety of algorithms [19].

Despite the considerable volume of research on evolutionary multiobjective optimization (see for example [19, 24, 38, 13, 119]), until recently, little emphasis had been placed on using efficiency as the main design goal when proposing new multiobjective evolutionary algorithms. Another important issue generally omitted in the current literature on evolutionary multiobjective optimization is on-line adaptation (i.e., the use of mechanisms, based on information gathered during the evolutionary process, for adjusting the algorithm's parameters).

1.2 Motivation

The main motivation of this work was precisely to design and implement new multiobjective evolutionary algorithms in which these two aspects previously indicated were emphasized: efficiency (measured in terms of obtaining the highest quality results possible,

with the lowest amount of fitness function evaluations) and on-line adaption (i.e., to make it unnecessary to fine tune by hand the parameters required by the algorithms).

Our research will focus particularly on particle swarm optimization, although some of the concepts proposed may be applicable to other multiobjective evolutionary algorithm.

1.3 Objectives

The main objectives of this work are the following:

- To gain deep knowledge on evolutionary multiobjective optimization using particle swarm optimization.
- To gain deep knowledge on techniques to maintain diversity in the context of evolutionary multiobjective optimization (mainly focused on the use of particle swarm optimization).
- To design and implement mechanisms for self- or on-line adaptation in a multiobjective particle swarm optimization algorithm. The main idea is to produce an approach whose parameters do not require any manual fine-tuning. Efficiency is another important design guideline for the aforementioned algorithm.
- To validate the proposed algorithm, comparisons should be performed with respect to approaches representative of the state-of-the-art in the area, using standard metrics and test functions.

1.4 Contributions

The expected contributions derived from this research work are the following:

- At least one new multiobjective evolutionary algorithm based on particle swarm optimization that doesn't require a manual fine tuning of its parameters and whose performance (measured both in terms of efficiency and quality of the results achieved) is competitive with respect to algorithms representative of the state-of-the-art in the area.
- At least one new mechanism to maintain diversity in a multiobjective particle swarm optimization algorithm.

1.5 Document Outline

The remainder of this document is organized as follows. Chapter 2, introduces some basic concepts necessary to understand the rest of the document and the metrics usually adopted to allow a quantitative assessment of the performance of multiobjective optimization algorithms. A brief description of evolutionary algorithms is presented in Chapter 3. Chapter 4 presents the main mathematical programming techniques for multiobjective optimization and shows the state-of-the-art in evolutionary computation for multiobjective

optimization. Chapter 5 presents an extension of the heuristic called "particle swarm optimization" (PSO) that is able to deal with multiobjective optimization problems. The proposed approach uses the concept of Pareto dominance to determine the flight direction of a particle and is based on the idea of having a set of sub-swarms instead of single particles. In each sub-swarm, a PSO algorithm is executed and, at some point, the different sub-swarms exchange information. Our proposed approach is validated using several test functions taken from the evolutionary multiobjective optimization literature and is compared with respect to algorithms representative of the state-of-the-art in the area. Chapter 6, presents a simple mechanism to handle constraints with a particle swarm optimization algorithm. Our proposal uses a simple criterion based on closeness of a particle to the feasible region in order to select a leader. Additionally, this constraint-handling mechanism is incorporated to a multiobjective particle swarm. In Chapter 7, we discuss self adaptation in the context of multiobjective optimization. We propose a revised version MOPSO (developed in Chapter 5) for multiobjective optimization which does not require any parameter fine-tuning. The new approach is validated using several test function and metrics taken from the specialized literature and it is compared with respect to two algorithms representative of the state-of-the-art in the area. Finally, Chapter 8, gives conclusions an future trends.

2

Background

2.1 Introduction

THE most important aim of this chapter is to introduce the basic concepts, definitions and performance measures related to multiobjective optimization.

We begin by laying a conceptual and theoretical basis for both global and multiobjective optimization in Section 2.2. Then, we introduce several concepts and definitions related to multiobjective optimization in Section 2.3. Next, a brief review of performance measures commonly used in evolutionary multiobjective optimization is given in Section 2.4. Finally, definitions of several test functions taken from the specialized literature are shown.

2.2 Optimization

Many optimization problems are very complex and difficult to solve by the use of simple common sense or by intuition. Engineers usually have to deal with problems of minimizing or maximizing[1] one or many objectives.

2.2.1 Global Optimization

The general (single-objective) global optimization problem can be stated as follows:

Definition 1 (Global optimization):

$$\text{Find } \vec{x} \text{ which optimizes } f(\vec{x}) \tag{2.1}$$

subject to:

[1] Without loss of generality, in this document we will refer only to minimization problems.

$$g_i(\vec{x}) \leq 0, \quad i = 1, \ldots, n \tag{2.2}$$

$$h_j(\vec{x}) = 0, \quad j = 1, \ldots, p \tag{2.3}$$

where $\vec{x}$ is the vector of solutions $\vec{x} = [x_1, x_2, \ldots, x_r]^T$, n is the number of inequality constraints and p is the number of equality constraints (in both cases, constraints could be linear or nonlinear). □

2.2.2 Multiobjective Optimization

Multiobjective optimization can be defined as the problem of finding [92]:

> *a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence, the term "optimize" means finding such a solution which would give the values of all the objective functions acceptable to the decision maker*

The above definition can be stated in a formal way as follows:

Definition 2 (General Multiobjective Optimization Problem (MOP)): *Find the vector $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ which will satisfy the m inequality constraints:*

$$g_i(\vec{x}) \leq 0 \qquad i = 1, 2, \ldots, m \tag{2.4}$$

the p equality constraints

$$h_i(\vec{x}) = 0 \qquad i = 1, 2, \ldots, p \tag{2.5}$$

and will optimize the vector function

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T \tag{2.6}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables. □

In other words, we wish to determine from among the set $\mathcal{F}$ (the feasible region) of all numbers which satisfy (2.4) and (2.5) the particular set $x_1^*, x_2^*, \ldots, x_n^*$ which yields the optimum values of all the k objective functions of the problem.

2.3 Background Concepts

Decision variables: The decision variables are a set of n parameters whose values give a solution (can be valid or not) to a problem. These parameters are denoted as $x_j,\ j = 1, 2, \ldots, n$. In this work, these variables will be represented by:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.7}$$

The same can also be written as:

$$\vec{x} = [x_1, x_2, \ldots, x_n]^T \tag{2.8}$$

Constraints: Most real world optimization problems have (natural and problem dependant) constraints to be satisfied (they draw up the boundaries of the feasible set). Constraints are functions of the decision variables and can be expressed in form of mathematical inequalities (eq.(2.9)) or equalities (eq (2.10)).

$$g_i(\vec{x}) \leq 0, \ i = 1, \ldots, m \tag{2.9}$$

$$h_i(\vec{x}) = \ 0, \ i = 1, \ldots, p \tag{2.10}$$

For an inequality constraint that satisfies $g_i(\vec{x}) = 0$, then we will say that it is active at $\vec{x}$. All equality constraints h_j (regardless of the value of $\vec{x}$ used) are considered active at all points of $\mathcal{F}$.

Objective functions: The objective functions are the evaluation criteria used to estimate how good a solution is. As in the case of the constraints, objective functions are functions of the decision variables. In multiobjective optimization problems there are k (≥ 2) objective functions ($f_1(\vec{x}), \ f_2(\vec{x}), \ldots, \ f_k(\vec{x})$). In this document,we will represent $\vec{f}$ in the following way:

$$\vec{f}(\vec{x}) = \begin{bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ \vdots \\ f_k(\vec{x}) \end{bmatrix} \tag{2.11}$$

The same can also be written as:

$$\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T \tag{2.12}$$

Pareto dominance:

Definition 3 (Pareto Dominance): *A vector $\vec{x}^* = (x_1, \ldots, x_k)$ is said to dominate $\vec{y}^* = (y_1, \ldots, y_k)$ (denoted by $\vec{x} \preceq \vec{y}$) if and only if x is partially less than y, i.e., $\forall i \in \{1, \ldots, k\}, \ x_i \leq y_i \wedge \exists i \in \{1, \ldots, k\} : x_i < y_i$.* □

As an example, for the case of 2 decision vectors $\vec{x}^*$, $\vec{y}^* \in X$,

$\vec{x}^* \prec \vec{y}^*$	$iff\ f_i(\vec{x}^*) < f_i(\vec{y}^*)$
$\vec{x}^*$ strictly dominates $\vec{y}^*$	for every $i = 1, ..., k$
$\vec{x}^* \preceq \vec{y}^*$	$iff\ f_i(\vec{x}^*) \leq f_i(\vec{y}^*)$
($\vec{x}^*$ weakly dominates $\vec{y}^*$)	for every $i = 1, ..., k$
$\vec{x}^* \sim \vec{y}^*$	$iff\ f_i(\vec{x}^*) \not\leq f_i(\vec{y}^*) \wedge f_i(\vec{y}^*) \not\leq f_i(\vec{x}^*)$
($\vec{x}^*$ is incomparable[2] to $\vec{y}^*$)	for every $i = 1, ..., k$

These definitions are analogous for maximization problems ($\succ, \succeq, \sim$).

In Figure 2.1 we can see the difference between decision variable space and objective function space.

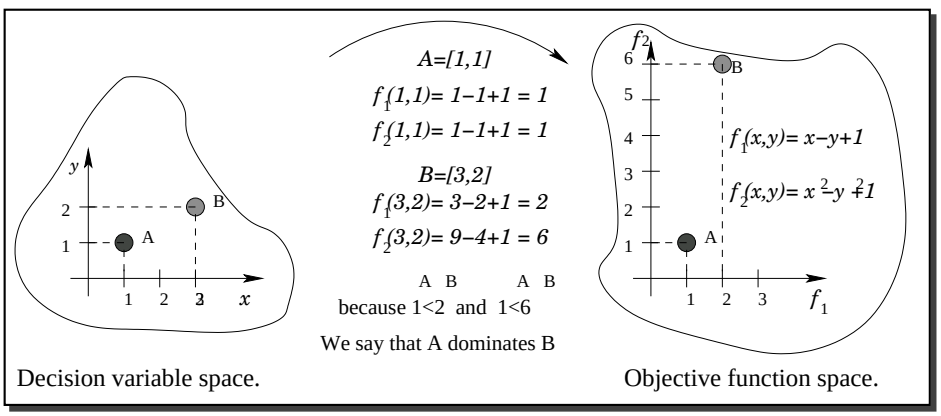

Figure 2.1: Example that is meant to show the difference between decision variable space (left) and objective function space (right).

To reinforce those concepts, let's consider the following example:
$\vec{x}^* = [2.4, 5.3, 4.5]^T$, $\vec{y}^* = [2.4, 5.3, 4.8]^T$ and $\vec{z}^* = [3.4, 5.4, 4.7]^T$

$\vec{x}^* \prec \vec{z}^*$ because $2.4 < 3.4$, $5.3 < 5.4$ and $4.5 < 4.7$,
$\vec{x}^* \preceq \vec{y}^*$ because $2.4 \leq 2.4$, $5.3 \leq 5.3$ and $4.5 < 4.8$ and
$\vec{y}^* \sim \vec{z}^*$ because $2.4 \leq 3.4$, $5.3 \leq 5.4$ and $4.8 > 4.7$.

Pareto optimal set:

Definition 4 (Pareto optimal set): *The Pareto optimal set (P^*) can be defined as:*

$$P^* = \{\vec{x}^* \in \mathcal{F} \mid \neg\exists\ \vec{y}^* \in \mathcal{F} \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{x}^*)\} \tag{2.13}$$

□

[2] $\vec{x}$ and $\vec{y}$ are non-dominated vectors between themselves.

In words, we can say that a decision vector which belongs to the feasible set $\mathcal{F}$ is a Pareto optimum if there is no other decision vector $\vec{y}^*$ which belongs to $\mathcal{F}$ and dominates it.

The Pareto optimal set is defined in decision variable space.

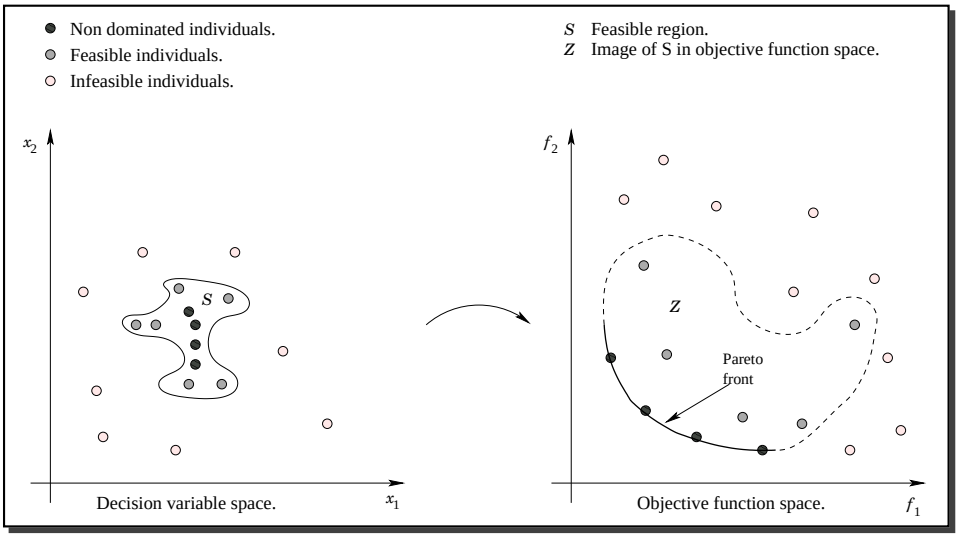

Figure 2.2: Example that is meant to show the difference between the Pareto optimal set (left) and the Pareto front (right).

When plotted in objective space, the non-dominated vectors are collectively known as the *Pareto front*. A graphical illustration of the Pareto optimal set (i.e., decision variable space) and the Pareto front (i.e., objective function space) of a problem is provided in Figure 2.2. In the general case, it is impossible to find an analytical expression of the line or surface that contains these points. The normal procedure to generate the Pareto front is to compute the feasible points $\mathcal{F}$ and their corresponding $f(\mathcal{F})$. When there is a sufficient number of these, it is then possible to determine the non-dominated points and to produce the Pareto front.

Some problems present fronts different from the true Pareto front which attract most of the solutions. They are known as false (or local) Pareto fronts. Figure 2.3 is meant to denote the difference between a false and a true Pareto front.

ϵ-dominance ϵ-dominance is a relaxed form of dominance. It is defined as follows:

Definition 5 (ϵ-dominance): *A vector $\vec{x}^* = (x_1, \ldots, x_k)$ is said to ϵ dominate $\vec{y}^* = (y_1, \ldots, y_k)$ (denoted by $\vec{x} \preceq_\epsilon \vec{y}$) if and only if $(1+\epsilon) \cdot f(\vec{x} \leq f(\vec{y})$, $\forall i \in \{1, \ldots, k\}$.* □

ϵ-approximate Pareto set:

Definition 6 (ϵ-approximate Pareto set): *A decision vector $\vec{x}^*$ is an ϵ-approximate Pareto set if and only if:*

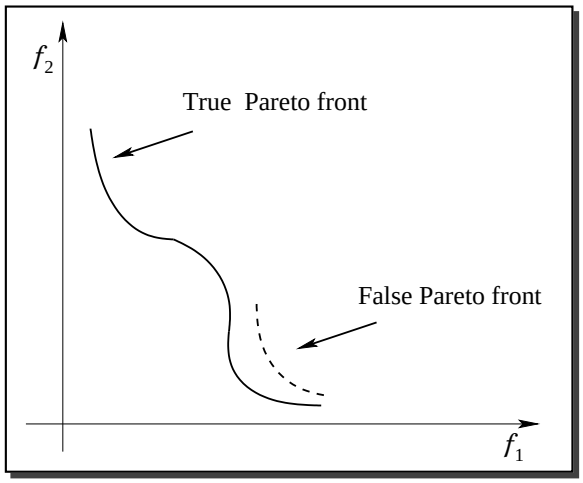

Figure 2.3: Example that is meant to show the difference between a false and a true Pareto front

$$\vec{x}^* \in \mathcal{F} \mid \neg\exists\, \vec{y}^* \in \mathcal{F} \mid \vec{f}(\vec{y}^*) \preceq_\epsilon \vec{f}(\vec{x}^*) \tag{2.14}$$

□

2.4 Performance Measures

In order to allow a quantitative assessment of the performance of an evolutionary multiobjective optimization algorithm, three issues are normally taken into consideration [132]:

1. Minimize the distance of the Pareto front produced by our algorithm with respect to the Pareto front (assuming we know its location).

2. Maximize the spread of solutions found, so that we can have a distribution of vectors as smooth and uniform as possible.

3. Maximize the number of elements of the Pareto optimal set found.

Based on these notions, we adopted the following performance measures to allow a quantitative comparison of results:

1. **Success Counting (SC)**: We define this measure based on the idea of the measure called *Error Ratio* proposed by Van Veldhuizen [117] to indicate the percentage of solutions (from the non-dominated vectors found so far) that are not members of the true Pareto optimal set. In this case, we count the number of vectors in the current set of non-dominated vectors available that are members of the Pareto optimal set:

$$SC = \sum_{i=1}^{n} s_i,$$

where n is the number of vectors in the current set of non-dominated vectors available; $s_i = 1$ if vector i is a member of the Pareto optimal set, and $s_i = 0$ otherwise. It should then be clear that $SC = n$ indicates an ideal behavior, since it would mean that all the vectors generated by our algorithm belong to the true Pareto optimal set of the problem. For a fair comparison, when we use this measure, all the algorithms should be encouraged to obtain their final Pareto fronts with the same number of vectors.

2. **Inverted Generational Distance (IGD)**: The concept of generational distance was introduced by Van Veldhuizen & Lamont [118, 120] as a way of estimating how far are the elements in the Pareto front produced by our algorithm from those in the true Pareto front of the problem. This measure is defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{2.15}$$

where n is the number of non-dominated vectors found by the algorithm being analyzed and d_i is the Euclidean distance (measured in objective space) between each of these and the nearest member of the true Pareto front. It should be clear that a value of $GD = 0$ indicates that all the elements generated are in the true Pareto front of the problem. Therefore, any other value will indicate how "far" we are from the global Pareto front of our problem. In our case, we implemented an "inverted" generational distance measure (IGD) in which we use as a reference the true Pareto front, and we compare each of its elements with respect to the front produced by an algorithm. In this way, we are calculating how far are the elements of the true Pareto front, from those in the Pareto front produced by our algorithm.

3. **Spacing** (SP): Here, one desires to measure the spread (distribution) of vectors throughout the non-dominated vectors found so far. Since the "beginning" and "end" of the current Pareto front found are known, a suitably defined metric judges how well the solutions in such front are distributed. Schott [108] proposed such a metric measuring the range (distance) variance of neighboring vectors in the non-dominated vectors found so far. This metric is defined as:

$$S \triangleq \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\overline{d} - d_i)^2}\ , \tag{2.16}$$

where $d_i = \min_j(\mid f_1^i(\vec{x}) - f_1^j(\vec{x}) \mid + \mid f_2^i(\vec{x}) - f_2^j(\vec{x}) \mid)$, $i, j = 1, \ldots, n, i \neq j$,$\overline{d}$ is the mean of all d_i, and n is the number of non-dominated vectors found so far. A value of zero for this metric indicates all members of the Pareto front currently available are equidistantly spaced.

4. **Two Set Coverage (TSC)**: This metric was proposed in [132], and it can be termed *relative coverage comparison of two sets*. Consider X', X'' as two sets of objective

function's vectors. TSC is defined as the mapping of the ordered pair (X', X'') to the interval $[0, 1]$:

$$TSC(X', X'') \triangleq \frac{|\{a'' \in X''; \exists a' \in X' : a' \succeq a''\}|}{|X''|} \tag{2.17}$$

If all points in X' dominate or are equal to all points in X'', then by definition $TSC = 1$. $TSC = 0$ implies the opposite. In general, $TSC(X', X'')$ and $TSC(X'', X')$ both have to be considered due to set intersections not being empty. Of course, this metric can be used for both spaces (objective function or decision variable space), but in this case we applied it in objective function space.

5. **Improved Generational Distance (ImGD)**: This metric is proposed in this thesis. Consider X', X'' as two sets of objective function's vectors. Let $d(x, Y)$ be the distance from point x to the set Y:

$$d(x, Y) = \min_{y \in Y} d(x, y)$$

where $d(x, y)$ refers to an Euclidean distance. The ImGD metric is defined as the mapping of (X', X'') to the ordered pair (x', x'')

$$ImGD(X', X'') = (x', x'') \tag{2.18}$$

such that:

$$x' = \frac{1}{n}\left[\sum_{x_i \in X'} \mathbf{d}(x_i, T) + \sum_{i=1}^{n} \mathbf{d}(t_i^{''}, X')\right] \tag{2.19}$$

$$x'' = \frac{1}{n}\left[\sum_{x_i \in X''} \mathbf{d}(x_i, T) + \sum_{i=1}^{n} \mathbf{d}(t_i^{'}, X'')\right] \tag{2.20}$$

where $t_i^{'}(t_i^{''}) \in T$ is the closest point from the true Pareto front to the point i from the set $X'(X'')$. If the result obtained is $(0, v)$ (where v is a positive value $\neq 0$), it would mean that the first expression (whose solutions are contained in X') generated all its solutions exactly on the true Pareto front, and these solutions also cover the region found by the second expression (whose solutions are contained in X''). The opposite holds as well. The main motivation of this metric was to overcome the main limitations of generational distance when measuring closeness to the true Pareto front of a problem. As we will see later on, in several cases in which generational distance provides some misleading results, this improved generational distance returns values that reflect, in a more accurate way, the behavior of each algorithm compared.

6. **Two Set Difference Hypervolume (HV)** This measure was proposed in [130]. Consider X', X'' as two sets of phenotype decision vectors. HV is defined by:

$$HV(X', X'') = \delta(X' + X'') - \delta(X'')$$

where the set $X' + X''$ is defined as the non-dominated vectors obtained from the union of X' and X'', and δ is the unary hypervolume measure. $\delta(X)$ is defined as the hypervolume of the portion of the objective space that is dominated by X. In this way, $HV(X', X'')$ gives the hypervolume of the portion of the objective space that is dominated by X' but not for X''.

Observe that since this metric is not simetric, then is necessary to consider both $HV(X', X'')$ and $HV(X'', X')$.

It should be clear that the values $HV(X', X'') = 0$ and $HV(X'', X') > 0$, indicate that all the elements of X' are dominated by the elements of X''. Therefore, $HV(X', X'') > HV(X'', X')$ indicates that X' *is better than* X''.

2.5 Test Functions

We present several test functions taken from the specialized literature below. Such test functions were selected because each one presents a different complexity.

2.5.1 Deb 1's Test Function

This example is a bi-objective test function proposed by Deb [23]:

$$\text{Minimize } f_1(x_1, x_2) = x_1 \tag{2.21}$$

$$\text{Minimize } f_2(x_1, x_2) = g(x_1, x_2) \cdot h(x_1, x_2) \tag{2.22}$$

where:

$$g(x_1, x_2) = 11 + x_2^2 - 10 \cdot \cos(2\pi x_2) \tag{2.23}$$

$$h(x_1, x_2) = \begin{cases} 1 - \sqrt{\frac{f_1(x_1,x_2}{g(x_1,x_2}} & \text{if } f_1(x_1, x_2) \leq g(x_1, x_2) \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

and $0 \leq x_1 \leq 1$, $-30 \leq x_2 \leq 30$.

Figure 2.4 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

2.5.2 Deb 2's Test Function

This example is a bi-objective test function proposed by Deb [23]:

$$\text{Minimize } f_1(x_1, x_2) = x_1 \tag{2.25}$$

$$\text{Minimize } f_2(x_1, x_2) = \frac{g(x_2)}{x_1} \tag{2.26}$$

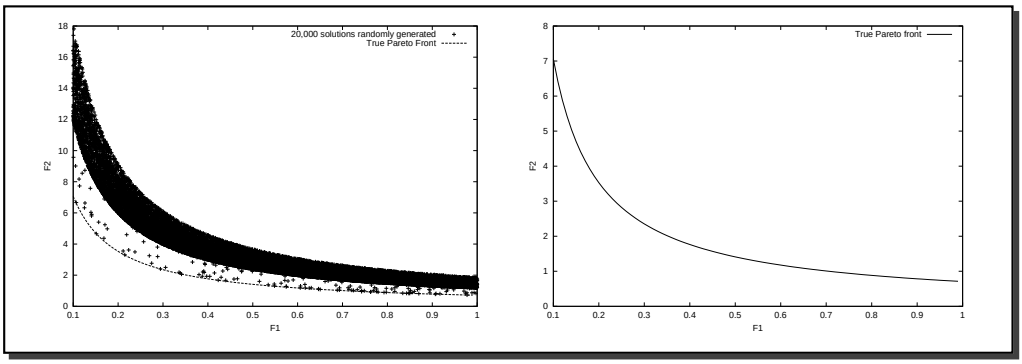

Figure 2.4: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Deb 1's test function.

$$g(x_2) = 2.0 - \exp\left\{-\left(\frac{x_2 - 0.2}{0.004}\right)^2\right\} - 0.8\exp\left\{-\left(\frac{x_2 - 0.6}{0.4}\right)^2\right\} \tag{2.27}$$

and $0.1 \leq x_1 \leq 1.0$, $0.1 \leq x_2 \leq 1.0$.

Figure 2.5 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

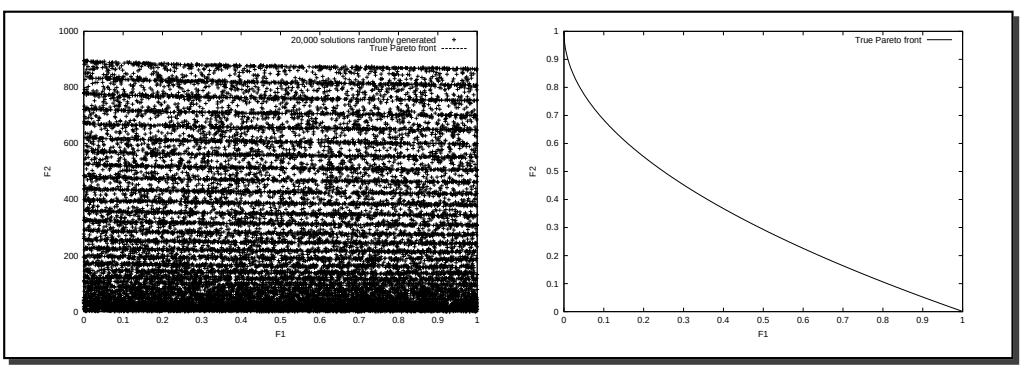

Figure 2.5: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Deb 2's test function.

2.5.3 Kursawe's Test Function

This test function was proposed by Kursawe [77]:

$$\text{Minimize } f_1(\vec{x}) = \sum_{i=1}^{n-1}\left(-10\exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2}\right)\right) \tag{2.28}$$

$$\text{Minimize } f_2(\vec{x}) = \sum_{i=1}^{n}\left(|x_i|^{0.8} + 5\sin(x_i)^3\right) \tag{2.29}$$

where: $-5 \leq x_1, x_2, x_3 \leq 5$

This problem has the Pareto front and the Pareto optimal set disconnected. Figure 2.6 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

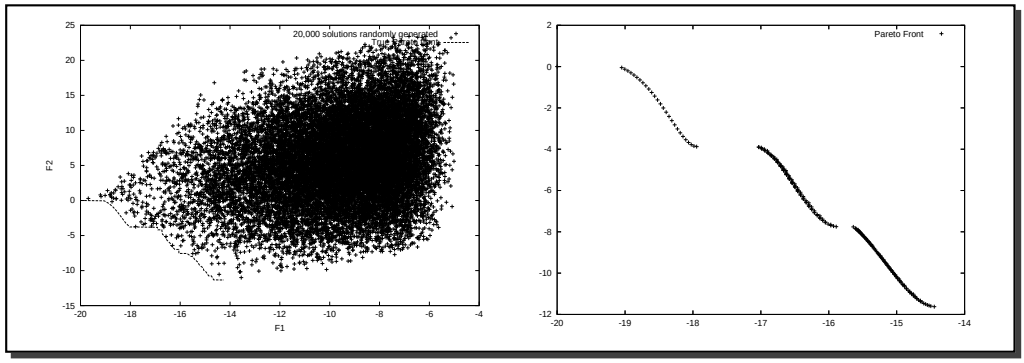

Figure 2.6: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Kursawe's test function.

2.5.4 DTLZ2's Test Function

Proposed by *Deb et al* in [29]. The test problem presents a big search space and $11^5 - 1$ local attractors.

Minimize:
$\vec{f_1} = (1 + g(x_M))(cos(x_1\pi/2))(cos(x_2\pi/2))...(cos(x_{M-2}\pi/2))(cos(x_{M-1}\pi/2))$,
$\vec{f_2} = (1 + g(x_M))(cos(x_1\pi/2))(cos(x_2\pi/2))...(cos(x_{M-2}\pi/2))(sen(x_{M-1}\pi/2))$,
$\vec{f_3} = (1 + g(x_M))(cos(x_1\pi/2))(cos(x_2\pi/2))...(sen(x_{M-2}\pi/2))$,

$$\vdots \qquad \vdots$$

$\vec{f_{M-1}} = (1 + g(x_M))(cos(x_1\pi/2))(sin(x_2\pi/2))$,
$\vec{f_M} = (1 + g(x_M))(sin(x_1\pi/2))$.

with:

$$\begin{aligned} g(x_M) &= \sum_{x_i \in X_M} (x_i - 0.5)^2 \\ M &= 3, \\ k &= 10 \\ n &= M + k - 1 \end{aligned}$$

y:

$$0 \leq x_i \leq 1 \quad \forall \quad i = 1, 2, ..., n$$

Figure 2.7 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

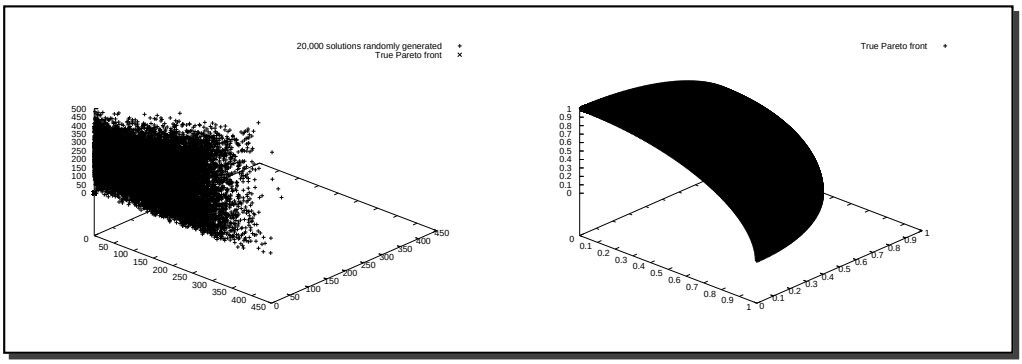

Figure 2.7: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for DTLZ2's test function.

2.5.5 ZDT1's Test Function

This test function was proposed by *Zitzler et al* in [135].

$$
\begin{aligned}
\text{Minimize} \quad & (f_1(\vec{x}), f_2(\vec{x})) \\
f_1(\vec{x}) &= x_1 \\
f_2(\vec{x}) &= g(\vec{x})h(f_1, g) \\
g(\vec{x}) = 1 &+ 9\sum_{i=2}^{m} x_i/(m-1), h(f_1, g) = 1 - \sqrt{f_1/g}
\end{aligned} \tag{2.30}
$$

where $m = 30$, and $x_i \in$[0,1].

Figure 2.8 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

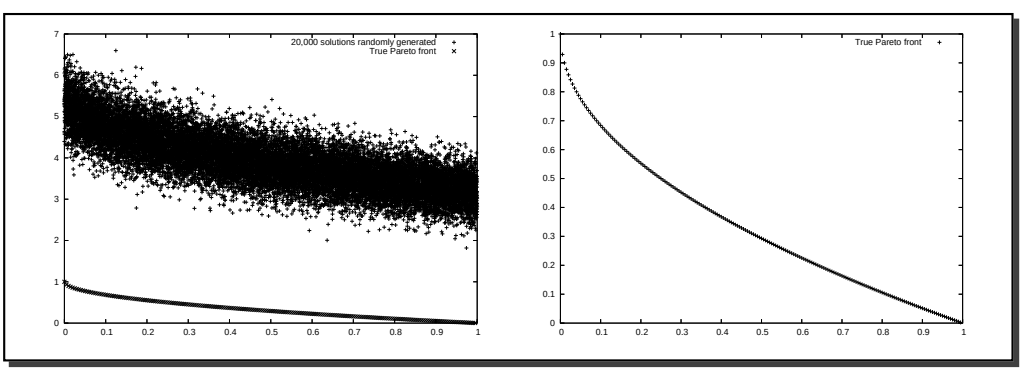

Figure 2.8: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT1's test function.

2.5.6 ZDT2's Test Function

This test function was proposed by *Zitzler et al* in [135].

$$
\begin{aligned}
\text{Minimize} \quad & (f_1(\vec{x}), f_2(\vec{x})) \\
f_1(\vec{x}) &= x_1 \\
f_2(\vec{x}) &= g(\vec{x})h(f_1, g) \\
g(\vec{x}) = 1 &+ 9\sum_{i=2}^{m} x_i/(m-1), h(f_1, g) = 1 - (f_1/g)^2
\end{aligned} \tag{2.31}
$$

where $m = 30$, and $x_i \in$[0,1].

Figure 2.9 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

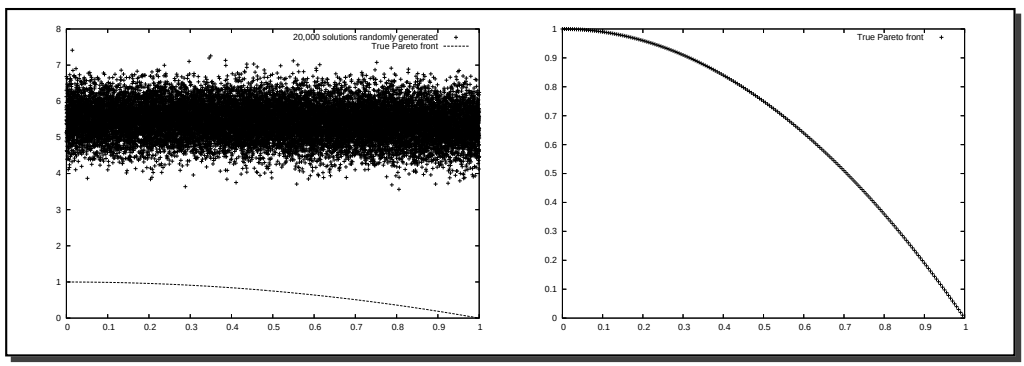

Figure 2.9: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT2's test function.

2.5.7 ZDT3's Test Function

This test function was proposed by *Zitzler et al* in [135].

$$
\begin{aligned}
\text{Minimize} \quad & (f_1(\vec{x}), f_2(\vec{x})) \\
f_1(\vec{x}) &= x_1 \\
f_2(\vec{x}) &= g(\vec{x})h(f_1, g) \\
g(\vec{x}) = 1 &+ 9\sum_{i=2}^{m} x_i/(m-1), h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1)
\end{aligned} \tag{2.32}
$$

where $m = 30$, and $x_i \in$[0,1].

Figure 2.10 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

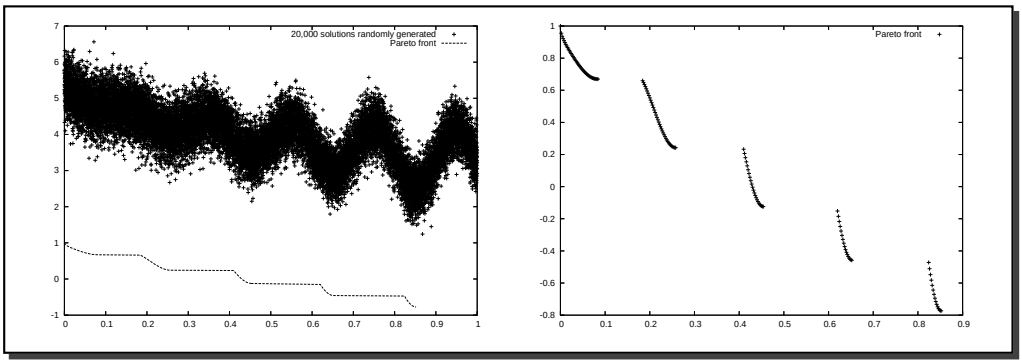

Figure 2.10: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT3's test function.

2.5.8 ZDT4's Test Function

This test function was proposed by *Zitzler et al* in [135].

$$
\begin{aligned}
\text{Minimize} \quad & (f_1(\vec{x}), f_2(\vec{x})) \\
f_1(\vec{x}) &= x_1 \\
f_2(\vec{x}) &= g(\vec{x})h(f_1, g) \\
g(\vec{x}) = 1 &+ 10(m-1) + \sum_{i=2}^{m}(x_i^2 - 10\cos(4\pi x_i), h(f1, g) = 1 - \sqrt{(f_1/g)}
\end{aligned} \tag{2.33}
$$

where $m = 10$, $x_0 \in$[0,1], and $x_i \in$[-5,5].

Figure 2.11 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

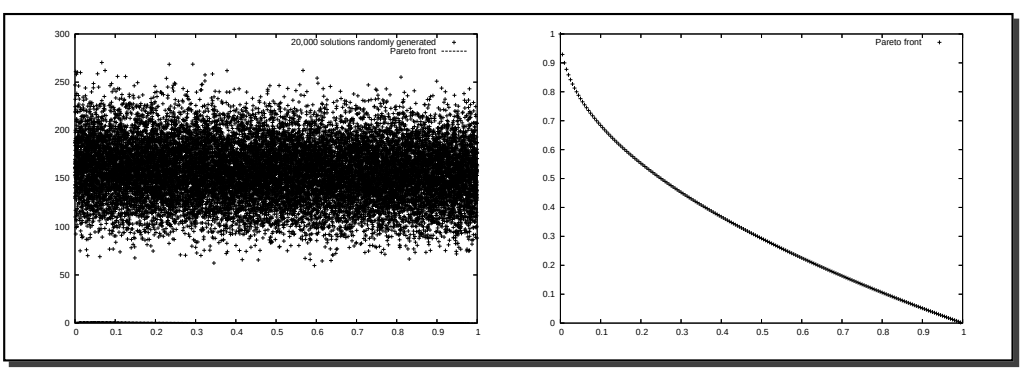

Figure 2.11: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT4's test function.

2.5.9 ZDT6's Test Function

This test function was proposed by *Zitzler et al* in [135].

$$
\begin{aligned}
\text{Minimize} \quad & (f_1(\vec{x}), f_2(\vec{x})) \\
f_1(\vec{x}) &= 1 - exp(-4x_1)\sin(6\pi x_1) \\
f_2(\vec{x}) &= g(\vec{x})h(f_1, g) \\
g(\vec{x}) = 1 &+ 9((\sum_{i=2}^{m} x_i)/(m-1))^{0.25}, h(f1, g) = 1 - (f_1/g)^2
\end{aligned} \tag{2.34}
$$

where $m = 10$, and $x_i \in$[0,1].

Figure 2.12 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

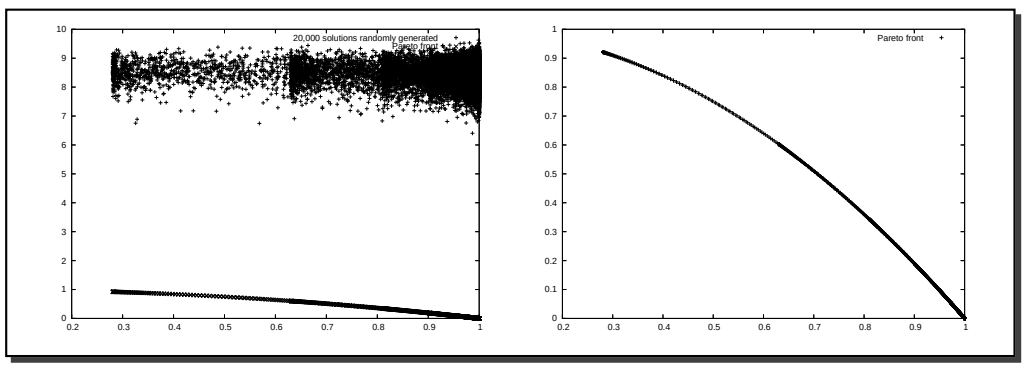

Figure 2.12: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT6's test function.

2.5.10 Kita's Test Function

This is a bi-objective optimization problem proposed by Kita [68]:

$$\text{Maximize} \quad f_1(x, y) = -x^2 + y \tag{2.35}$$

$$\text{Maximize} \quad f_2(x, y) = \frac{1}{2}x + y + 1 \tag{2.36}$$

subject to:

$$\frac{1}{6}x + y - \frac{13}{2} \leq 0 \tag{2.37}$$

$$\frac{1}{2}x + y - \frac{15}{2} \leq 0 \tag{2.38}$$

$$\frac{5}{x} + y - 30 \leq 0 \tag{2.39}$$

and $0 \leq x \leq 7.0$, $0 \leq y \leq 7.0$.

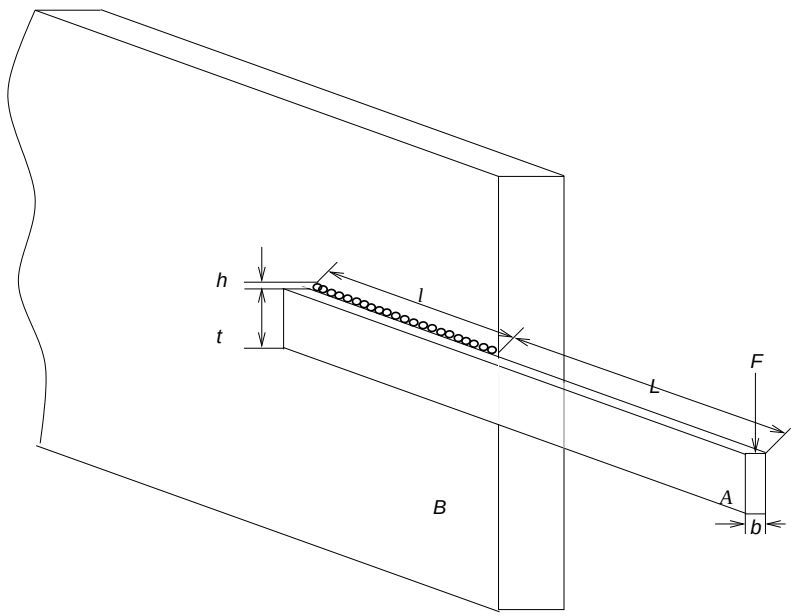

Figure 2.14: Welded Beam

This test function presents a concave Pareto front. Figure 2.13 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

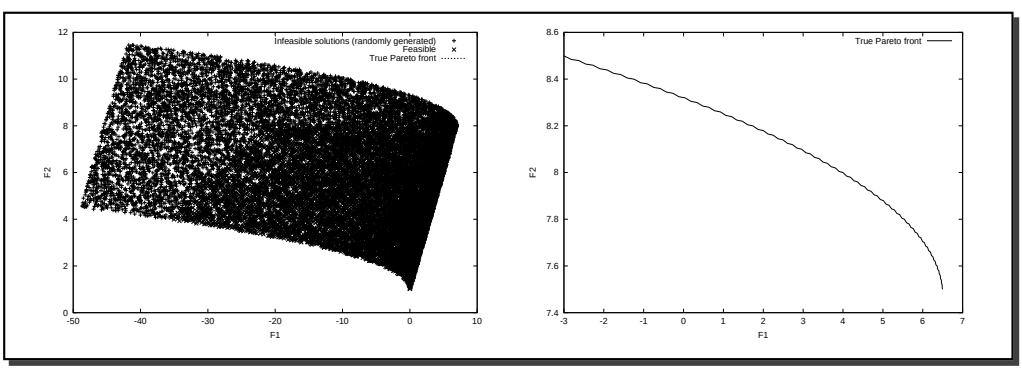

Figure 2.13: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for ZDT6's test function.

2.5.11 Welded Beam's Test Function

This example consists of the welded beam problem shown in Figure 2.14. The problem consists on [98] minimize the cost and minimize the end deflection beam subject to constraints on shear stress in weld (τ), bending stress in the beam (ρ), buckling load on the bar (P_c), and side constraints:

Design vector:

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} h \\ l \\ t \\ b \end{Bmatrix}$$

$$\text{Minimize} \quad f_1(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (L + x_2) \tag{2.40}$$

$$\text{Minimize} \quad f_2(\vec{x}) = (4FL^3)/(E x_3^3 x_4) \tag{2.41}$$

subject to:

$$g_1 : \ \tau \leq \tau_{max} \tag{2.42}$$

$$g_2 : \ \rho \leq \rho_{max} \tag{2.43}$$

$$g_3 : \ x_1 \leq x_4 \tag{2.44}$$

$$g_4 : \ x_1 \leq 0.125 \tag{2.45}$$

$$g_5 : \ F \leq P_c \tag{2.46}$$

where:

$$F = 6,000 \ lb$$

$$L = 14 \ in$$

$$E = 30 \times 10^6 \ psi$$

$$G = 12 \times 10^6 \ psi$$

$$\tau_{max} = 13,600 \ psi$$

$$\rho_{max} = 30,000 \ psi$$

$$\alpha = \frac{1}{(3G x_3 x_4^3)}$$

$$I = \frac{1}{12x_3x_4^3}$$

$$P_c = (4013\sqrt{EI\alpha}/L^2) \times (1 - (x_3/(2L) \times \sqrt{EI/\alpha}))$$

$$\rho = (6FL)/(x_4x_3^2)$$

$$J = 2 * (0.707x_1x_2(x_2^2/12 + ((x_1 + x_4)/2)^2))$$

$$R = \sqrt{(x_2^2)}/4 + ((x_1 + x_2)/2)^2$$

$$M = F(L + x_2/2)$$

$$cost = x_2/(2R)$$

$$\tau'' = MR/J$$

$$\tau' = F/(\sqrt{2} + x_1x_2)$$

$$\tau = \sqrt{(\tau'^2 + 2\tau'\tau''cost + \tau''^2)}$$

$$0 \leq x_1, x_4 \leq 2$$

$$0 \leq x_2, x_3 \leq 10$$

This test function presents a convex Pareto front. Figure 2.15 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

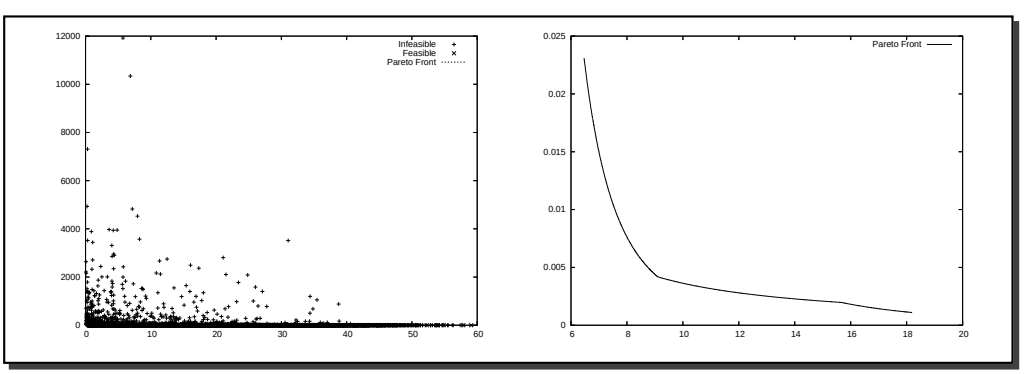

Figure 2.15: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Welded Beam's test function.

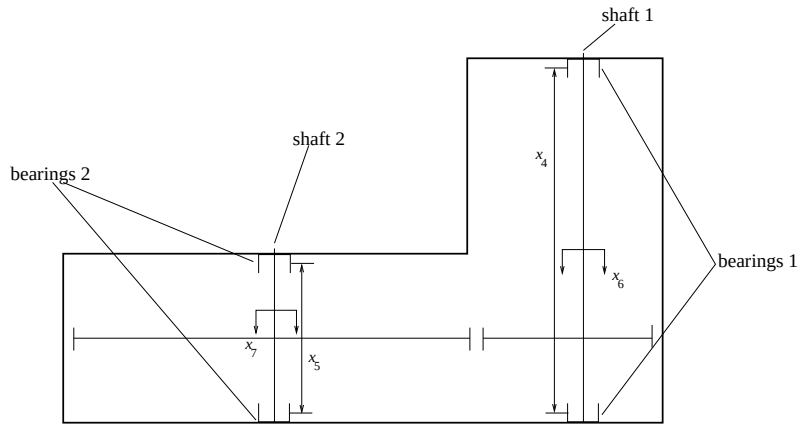

Figure 2.16: Plane truss used for the fourth example. The structural volume and the joint displacement (Δ) are to be minimized.

2.5.12 Speed Reducer's Test Function

This example consists of the speed reducer problem shown in Figure 2.16. The problem consists on minimize the stress and the weight of the speed reducer, considering the face width (x_1), the module of teeth (x_2), the number of teeth on pinion (x_3), the length of the shaft 1 between bearings (x_4), the length of the shaft 1 between bearings (x_5), the diameter of shaft 1 (x_6) and the diameter of shaft 2 (x_7).

This is defined as follows [98]:

Minimize

$$\begin{aligned} f_{weight} = \quad f_1(\vec{x}) = \quad & 0.7854x_1x_2^2(10x_3^2/3 + 14.933x_3 - 43.0934) - 1.508x_1(x_6^2 + x_7^2) \\ & +7.477(x_6^3 + x_7^3) + 0.7854(x_4x6^2 + x_5x7^2) \\ f_{stress} = \quad f_2(\vec{x}) = \quad & \frac{\sqrt{(745.0x_4/x_2x_3)^2 + 1.6910^7}}{0.1x_6^3} \end{aligned} \tag{2.47}$$

such that

$$g_1 : \ \frac{1.0}{x_1x_2^2x_3} - \frac{1.0}{27.0} \leq 0$$

$$g_2 : \ \frac{1.0}{x_1x_2^2x_3^2} - \frac{1.0}{397.5} \leq 0$$

$$g_3 : \ \frac{x_4^3}{x_2x_3x_6^4} - \frac{1.0}{1.93} \leq 0$$

$$g_4 : \ \frac{x5^3}{x_2x_3x_7^4} - \frac{1.0}{1.93} \leq 0$$

$$g_5 : \ x_2x_3 - 40.0 \leq 0$$

$$g_6: \; x_1/x_2 - 12.0 \leq 0$$

$$g_7: \; 5.0 - x_1/x_2 \leq 0$$

$$g_8: \; 1.9 - x_4 + 1.5x_6 \leq 0$$

$$g_9: \; 1.9 - x_5 + 1.1x_7 \leq 0$$

$$g_{10}: \; \frac{\sqrt{(745x_4/x_2x_3)^2} + 1.6910^7}{0.1x_6^3} \leq 1300)$$

$$g_{11}: \; \frac{\sqrt{(745x_5/x_2x_3)^2} + 1.57510^8}{0.1x_7^3} \leq 1100)$$

where:

$$2.6 \leq x_1 \leq 3.6$$

$$0.7 \leq x_2 \leq 0.8$$

$$17 \leq x_3 \leq 28$$

$$7.3 \leq x_4 \leq 8.3$$

$$7.3 \leq x_5 \leq 8.3$$

$$2.9 \leq x_6 \leq 3.9$$

$$5.0 \leq x_7 \leq 5.5$$

This test function presents a disconnected Pareto optimal set. Figure 2.17 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

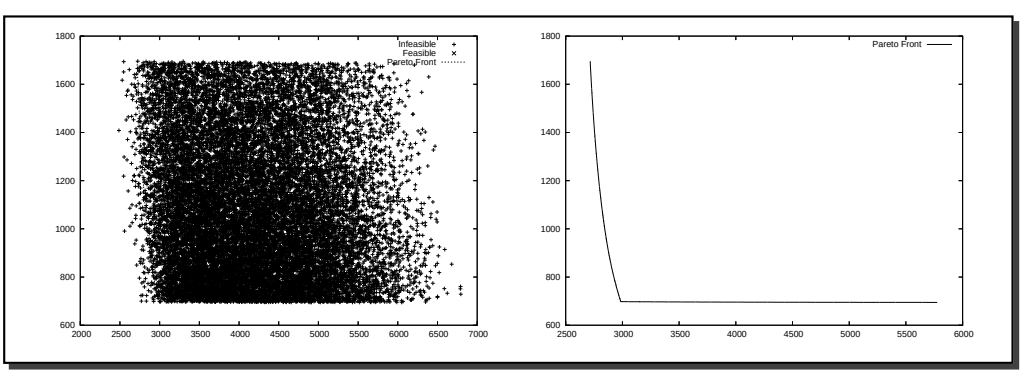

Figure 2.17: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Speed Reducer's test function.

2.5.13 Osyczka2 's Test Function

This example is a bi-objective test function proposed by Osyczka [93]:

$$\text{Maximize } f_1(\vec{x}) = 25(x_1-2)^2 + (x_2-2)^2 + (x_3-1)^2 + (x_4-4)^2 + (x_5-1)^2 \tag{2.48}$$

$$\text{Minimize } f_2(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 \tag{2.49}$$

such that

$$g_1 : \ (x_1 + x_2 - 2) \geq 0$$

$$g_2 : \ (6 - x_1 - x_2) \geq 0$$

$$g_3 : \ (2 - x_2 + x_1) \geq 0$$

$$g_4 : \ (2 - x_1 + 3 * x_2) \geq 0$$

$$g_5 : \ (4 - (x_3 - 3)^2 - x_4) \geq 0$$

$$g_6 : \ (x_5 - 3)^2 + x_6 - 4 \geq 0$$

where:

$$0 \leq x_1, x_2, x_6 \leq 10$$

$$1 \leq x_3, x_5 \leq 5$$

$$0 \leq x_4 \leq 6$$

This test function presents a disconnected Pareto optimal set. Figure 2.18 shows the graphical result produced by 20,000 solutions randomly generated (left side). The true Pareto front of the problem is shown as a continuous line (right side).

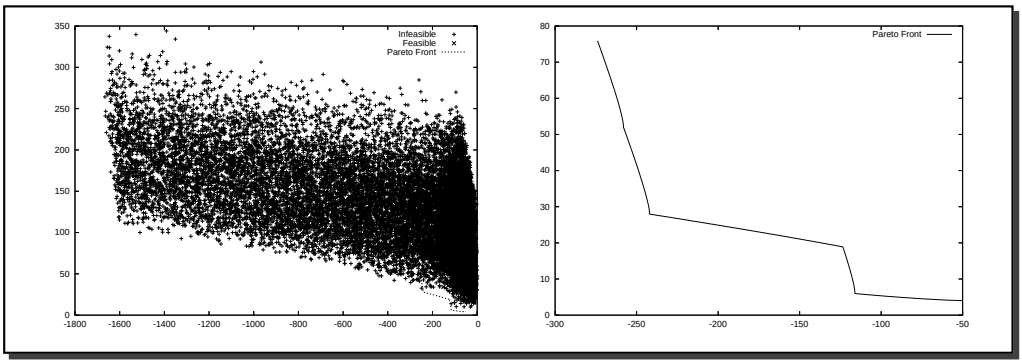

Figure 2.18: The graphic of the left shows 20,000 solutions randomly generated, and the graphic of the right shows the true Pareto front for Osyczka 2's test function.

3

Evolutionary Algorithms

3.1 Introduction

In this chapter, we will provide a description of evolutionary single/multiobjective optimization algorithms that is based on a unifying view presenting a general scheme that forms the common basis of all the current variations reported in the specialized literature.

The main components of an evolutionary algorithm are: an initialization process, a parent selection mechanism, variation operators and a survivor selection mechanism. Each of them will be briefly discussed in this chapter, and we will also explain their corresponding role in multiobjective evolutionary algorithms.

3.2 Evolutionary Algorithms

In nature, those individuals that are best adapted to their environment have more opportunities to *compete* for resources and *reproduce* [21]. In this process (usually known as *"the survival of the fittest"*), *natural selection* plays the main role. However, in [21], Darwin also identified that some rare modifications in the phenotype[1] (*mutations*) affect directly the performance of an individual in a given environment.

Natural evolution can be seen as an optimization problem, where the aim is to adapt best the species to their environment. Therefore, it is not surprising that scientists have taken inspiration from nature, since *Neo-Darwinism*[2] has been the most important model to explain natural evolution.

Neo-Darwinism points out that all the diversity of life in our planet can be explained through 4 processes:

[1]Phenotype: The realized expression of the genotype. Genotype: The sum of inherited characters maintained within the entire reproducing population [35].

[2]Neo-Darwinism is the name given to the fusion of natural selection theory [21], Mendel's inheritance laws [84] and Weismann's selectionism [123]

- Reproduction: This mechanism ensures the inheritance of genetic material from the current to the next generation.
- Mutation: It is produced if a copying error occurs in the genetic material (during reproduction). A mutation is beneficial to an organism if it produces a fitness increase in its adaptation to the environment.
- Competition: It is a natural process, in which the organisms have a continuous confrontation to survive and to inherit their genetic code to next generation.
- Selection: In an environment that can only host a limited number of individuals, only the organisms that compete most effectively for resources can survive and reproduce.

Evolutionary Algorithms (EA) are able to generate solutions for difficult real world problems, mainly because of their analogy with the Neo-Darwinism (particularly, the fact that in nature, the populations evolve through generations using the mechanism of natural selection and the survival of the fittest principle). The evolution of such solutions depends on the appropriate implementation of the following points:

- The encoding of the data structures used to represent the solutions of the problem. Each solution is known as an "individual" and a set of individuals is called a "population".
- Operators to modify the individual's traits (so-called variation operators).
- A fitness function which plays the role of the environment.
- The survival of the fittest is implemented through of the use of a selection procedure which plays the role of environmental pressure.

Nowadays, EAs are very popular in many disciplines, mainly because they present several advantages in optimization with respect to traditional techniques, such as:

- EAs do not need any specific knowledge about the problem. However if such knowledge is available, it can be easily incorporated.
- EAs are conceptually very simple.
- EAs have a wide applicability.
- EAs can easily exploit parallel architectures.
- EAs can usually adapt their own parameters.
- EAs are less susceptible to be trapped into a local minimum/maximum (since they are population-based techniques).
- EAs can cooperate with other search/optimization techniques.

Traditionally, EAs have been grouped into three main paradigms:

Algorithm 1 The general scheme of an EA

```
Initialize population with randomly generated solutions
Evaluate each solution
while stop condition is not reached do
    Select parents for reproduction
    Apply variation operators to the parents selected
    Evaluate new candidates
    Select the best individuals (among all) for the next generation (elitism)
end while
```

- Evolutionary Programming.
- Evolution Strategies.
- Genetic Algorithms.
  - Genetic Programming (it is a genetic algorithm variant).

Next we will briefly describe each of them.

3.2.1 Evolutionary Programming

Lawrence J. Fogel conceived the use of a form of simulated evolution to solve problems in the mid-1960s [36]. The technique that he developed was called Evolutionary Programming (EP). Intelligence in this technique can be seen as an adaptive behavior. This approach emphasizes the interactions between parents and offspring.

In *EP*, the crossover operator is not required, since *EP* is an abstraction of the evolutionary process at an species level (two different species can not be recombined). EP uses probabilistic selection (usually stochastic tournaments). Nowadays, there are several variants of this technique.

The basic evolutionary programming algorithm is shown in Algorithm 2.

Algorithm 2 Evolutionary Programming Algorithm

```
Initialize population with randomly generated solutions
repeat
    Apply mutation
    Evaluate new candidates
    Select those solutions which will be kept
    Generate the new population
until stop condition is reached
```

3.2.2 Evolution Strategies

In the mid-60s *Peter Bienert* [6], *Ingo Rechenberg* [101] and *Hans-Paul Schwefel* [109] developed a randomly discrete adjustment method, inspired in the mutation mechanism that exists in nature. The technique was called *Evolution Strategy* (EE) and it was initially used to solve highly complex hydrodynamical problems.

The original version is the so-called $(1+1)-EE$ and it uses a single parent which generates a single offspring. This offspring survives to become the parent at the following generation, only if it has a better fitness that his parent; otherwise it is eliminated (this is called extinctive selection).

In the $(1+1)-EE$, a new individual is generated using:

$$\overline{x}^{t+1} = \overline{x}^{t} + N(0, \sigma)$$

where t refers to the current generation (or iteration), and $N(0, \sigma)$ is a vector of Gaussian numbers with a zero mean and a standard deviation σ.

Evolution Strategies evolve not only the problem's variables, but also its own parameters (standard deviation), in a process called "self-adaptation". Recombination is possible but it is normally a secondary operator. The selection mechanism adopted is usually deterministic. Its general algorithm is shown in Algorithm 3.

Algorithm 3 Evolution Strategies Algorithm

```
Initialize the population G with randomly generated solutions
t := 0
Evaluate G(t)
repeat
    Select G_temp from G(t)
    Reproduce G_temp to generate G(t+1)
    Apply mutation to G(t+1)
    Evaluate G(t+1)
    Select survivors
    t := t+1
until stop condition is reached
```

3.2.3 Genetic Algorithms

Genetic algorithms (GAs) (originally called *reproductive plans*) were introduced by *John H. Holland* [53] in the early 1960s. Holland's main interest was to study natural adaptation in order to apply it to machine learning. Nowadays, GAs are the most popular type of evolutionary algorithm.

The general pseudocode of a simple GA [9] is shown in Algorithm 4.

Algorithm 4 Simple Genetic Algorithm

```
Initialize the population G with randomly generated solutions
t := 0
repeat
    Evaluate G(t)
    Select G_1(t) from G(t)
    Apply crossover and mutation to G_1(t) to generate G(t+1)
    t := t+1
until stop condition is reached
```

3.2.4 Genetic Programming

Nichal Lynn Cramer [20] and later, *John R. Koza* [74] proposed (in a different and independent way) the use of a tree representation in which a crossover operator for interchanging subtrees among different computer programs was implemented (with certain constraints imposed by the syntax of the programming language used).

The basic difference between both proposals is that Cramer used an interactive fitness function (i.e., the user had to supply (manually) the fitness value of each tree of the population), while Koza was able to automate it. Koza's proposal prevailed and nowadays is known as *Genetic Programming* (GP) [75].

The general algorithm of the GP [3] is shown in Algorithm 5.

Algorithm 5 Genetic Programming General Algorithm

1: Initialize the population G with randomly generated solutions
2: $t := 0$
3: **repeat**
4: Evaluate the programs of the population $G(t)$
5: Select $G_1(t)$ from $G(t)$
6: Apply genetic operators to $G_1(t)$ to generate $G(t+1)$
7: $t := t + 1$
8: **until** stop condition is reached

3.2.5 Swarm Intelligence

Swarm Intelligence (SI) encompasses a group of evolutionary optimization systems inspired by the collective behavior of social insect colonies and other animal societies. Such systems are made up by a population of simple agents interacting locally among them and with their environment. The aim of Swarm Intelligence researchers is the study of collective behavior in decentralized systems.

Particle Swarm Optimization and Ant Colony are two forms of SI. Instead of genetic operators, in these algorithms each individual (called agent in this context) varies itself according to its past experience and the local interaction with other agents. The interactions among agents usually cause a global behavior.

Since this thesis will be developed around Particle Swarm Optimization, an explanation of such approach is provided next.

3.2.5.1 Particle Swarm Optimization

Kennedy & Eberhart [65] proposed an approach called "Particle Swarm Optimization" (PSO) which was inspired on the choreography of a bird flock. Like other evolutionary algorithms, PSO uses a set of possible solutions which will be "evolving" until an optimal solution or a termination criteria is reached. In this case, each solution ($\vec{x}$) is represented by a particle. And, a particle swarm is a set of particles. The responsibility of evolving (moving) the swarm to the optimal region corresponds to the velocity equation. This equation is usually composed by three elements: a velocity inertia, a cognitive component

(*pbest*) and a social component (*gbest*). The entire approach can be seen as a distributed behavioral algorithm that performs (in its more general version) multidimensional search. In the simulation, the behavior of each particle is affected by either the best local (i.e., within a certain neighborhood) or the best global particle.

An interesting aspect of PSO is that it allows individuals to benefit from their past experiences (note that in other approaches such as the genetic algorithm, normally the current population is the only "memory" used by the individuals). PSO has been successfully used for both continuous nonlinear and discrete binary single-objective optimization [65].

The pseudocode of the PSO algorithm is shown in Algorithm 6. First, the particles are initially randomly initialized through the search space. These initial positions also initialize each particle's *pbest*. Next, the fittest particle from all the particle swarm is selected and assigned to the *gbest* solution. Then, the particle swarm flies the search space until certain termination criteria is reached. This flight consists on applying to the particle swarm a velocity equation, which updates the position and fitness of each particle. The new fitness obtained by each particle is compared with respect to the particle's *pbest* position; in case that the new position has a better fitness, then it replaces to the *pbest* position. The same procedure is performed for the *gbest* solution.

Algorithm 6 PSO Algorithm

1: $\vec{gbest} \leftarrow \vec{x_0}$
2: **for** $i = 0$ to $nparticles$ **do**
3: $\quad \vec{pbest_i} \leftarrow \vec{x_i} \leftarrow initialize_randomly()$
4: $\quad fitness_i \leftarrow f(\vec{x_i})$
5: $\quad$ **if** $fitness_i < f(\vec{gbest})$ **then**
6: $\quad\quad \vec{gbest} \leftarrow \vec{x_i}$
7: $\quad$ **end if**
8: **end for**
9: **repeat**
10: $\quad$ **for** $i = 0$ to $nparticles$ **do**
11: $\quad\quad$ **for** $d = 0$ to $ndimensions$ **do**
12: $\quad\quad\quad velocity_{id} \leftarrow W \times velocity_{id} + C_1 \times U(0,1) \times (pbest_{id} - x_{id}) + C_2 \times U(0,1) \times (gbest - x_{id})$
13: $\quad\quad\quad x_{id} \leftarrow x_{id} + velocity_{id}$
14: $\quad\quad$ **end for**
15: $\quad\quad fitness_i \leftarrow f(\vec{x_i})$
16: $\quad\quad$ **if** $fitness_i < f(\vec{pbest_i})$ **then**
17: $\quad\quad\quad \vec{pbest_i} \leftarrow \vec{x_i}$
18: $\quad\quad$ **end if**
19: $\quad\quad$ **if** $fitness_i < f(\vec{gbest}))$ **then**
20: $\quad\quad\quad \vec{gbest} \leftarrow \vec{x_i}$
21: $\quad\quad$ **end if**
22: $\quad$ **end for**
23: **until** Termination criterion

The PSO algorithm requires the following parameters:

- *Termination criterion*: it refers to the criterion adopted to conclude the execution of the algorithm (usually the total number of generations that the algorithm will be executed).

- *nparticles*: it refers to the total number of particles that will be over-flying the search space.
- W: it refers to the velocity inertia of the previous movement.
- $C1$: is the constant of the cognitive component. This constant indicates how strong will be the attraction from its best position.
- $C2$: is the social component. It indicates how strong will be the attraction from the best particle's position found so far.

3.3 Multiobjective Evolutionary Algorithms

There are several variants of multiobjective evolutionary algorithms (MOEAs). Recent MOEAs, however, can be classified into a general scheme. First, the initial population is filled by randomly generated solutions. Next, given the objective functions to be minimized, the population is evaluated and ranked on the basis of non-domination and distribution. Based on this rank, some of the best candidates are chosen to seed the next generation by applying variation operators. Next, the non-dominated solutions from both, the parents and offspring population are merged to become the parents for the next population. This process is repeated until a stop condition is reached.

The main difference of recent MOEAs with respect to earlier MOEAs is the use of elaborated forms of elitism (i.e. recent MOEAs spend a considerably part of their efforts on the selection of the best distributed non-dominated solutions and on mechanisms aimed to improve their convergence time). The aim of elitism is mainly to retain the non-dominated solutions generated during the search. The general pseudocode of a MOEA is given in Algorithm 7.

Algorithm 7 The general scheme of a MOEA

```
Initialize population with randomly generated solutions
Evaluate each solution
while stop condition is not reached do
    Select parents for reproduction (Pareto Selection Mechanism)
    Apply variation operators to the parents selected
    Evaluate new candidates
    Select the best individuals (among all) for the next generation (Survivor Selection Mechanism
    and retention of Non-Dominated Solutions)
end while
```

3.3.1 Multi-Objective Evolutionary Components

This Section will provide a description of mono/multiobjective evolutionary algorithms that is based on an unifying view presenting a general scheme that forms the common basis of most of the current variations reported in the specialized literature.

Initialization, parent selection mechanism, variation operators and survivor selection mechanism are the main components of an evolutionary algorithm [31]. We briefly discuss them below, and we explain their role in MOEAs.

3.3.2 Initialization

Like in single-objective EAs, the initial population is filled by randomly generated solutions (normally, a uniform distribution is adopted). The aim of this process is to spread solutions uniformly throughout the search space.

3.3.3 Parent Selection Mechanism

This mechanism is responsible for selecting the individuals who will become parents to give rise to the next generation based on their quality. The parent selection mechanism is typically probabilistic. Thus, non-dominated individuals have a higher probability of being selected. Nonetheless, dominated individuals are often given a small (but non-zero) chance of survival. There is evidence that indicates that the parent selection mechanism has a significant impact on the performance of a MOEA. For example in [97] it was discovered that the recombination of spatially dissimilar solutions tended to produce offspring that performed relatively badly, so it was necessary that the parent selection mechanism chose well-distributed but not spatially dissimilar solutions, for recombination purposes. The main drawback, if we pretend to select spatially similar solutions in the parent selection mechanism, is that it could be possible to lose promising points which are located far away from the majority. A good solution to solve this problem is to interact with an external archive, such that the algorithm takes the parents from here (this would avoid losing promising points). Furthermore, a good mechanism to keep diversity and to divide located solutions into clusters based on their spatial similarity is necessary. In this case, clustering techniques are needed.

A good clustering technique should be able to group the solutions with respect to their spatial similarity (as is shown in Figure 3.1).

Knowles [70] identified several features or dimensions of quality of mechanisms for promoting speciation and the maintenance of diversity:

1. Time complexity – particularly in terms of the population size.

2. Selection pressure and the exploration/exploitation tradeoff.

3. What measure of diversity is adopted: genotypic vs phenotypic in EAs, and different tradeoffs in objective space in MOEAs.

4. Accuracy and stability – how closely the method approaches the desired number of solutions on each optimum (usually related to the fitness of the optimum) and how steadily it maintains these numbers as selection continues i.e. in the steady state.

5. Robustness to optima of different sizes and shapes, and to optima distributed non-uniformly in the search space.

6. Parameterization/self-adaptiveness. How much *a priori* knowledge is needed about the size, position and number of optima in the solution space in order to apply the technique, how many parameters need to be fine-tuned, and how much robustness there is if parameters are not set accurately.

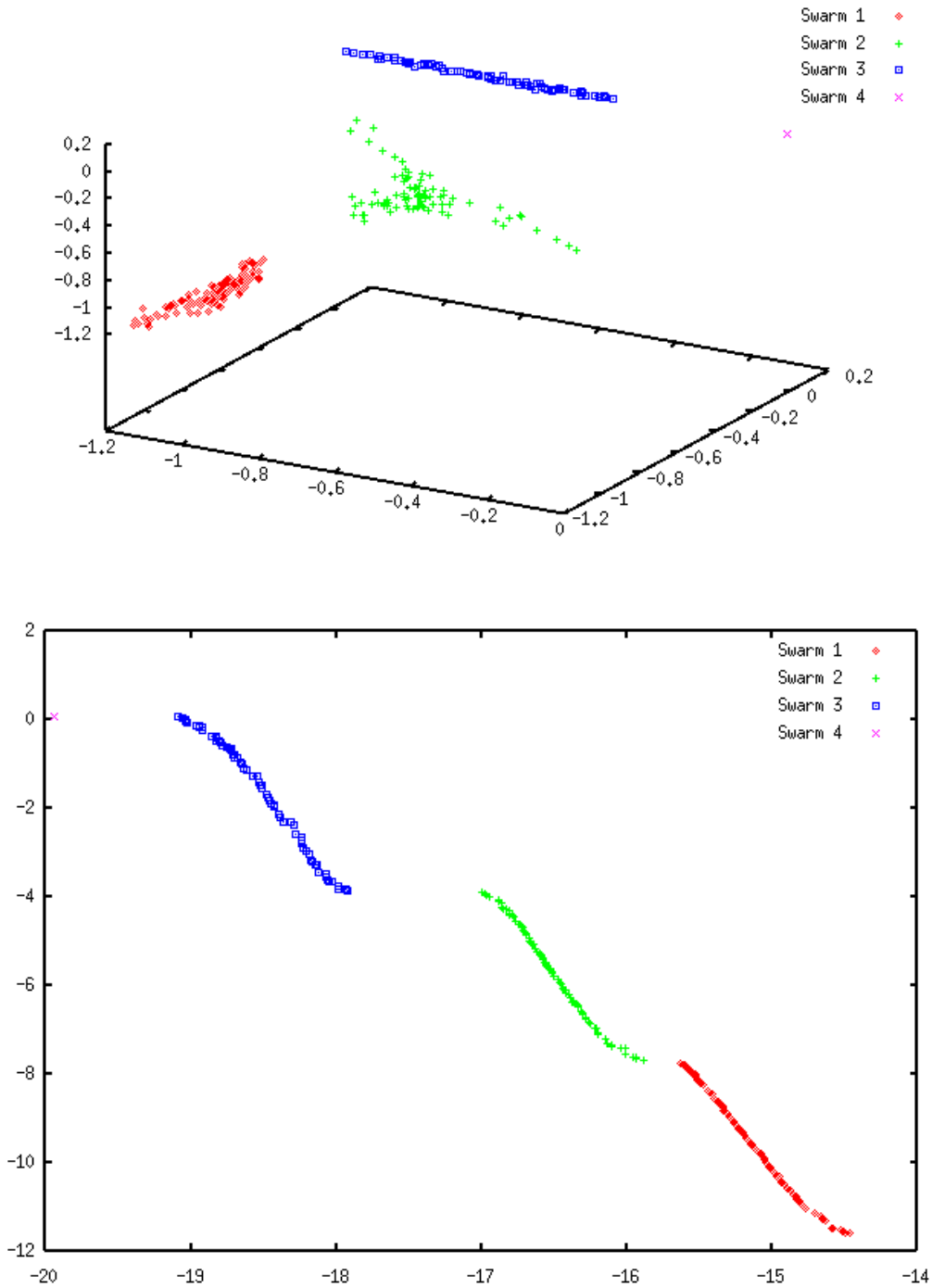

Figure 3.1: The top figure shows the location of 4 swarms clustered in the variable space of Kursawe's test function (see Section 2.5.3), and the figure at the bottom shows the same in the objective function space.

In addition to the previous features, the parent selection mechanism should include the following desirable characteristics:

- To interact with an external archive, with the aim of avoiding losing promissory points.
- To group and select spatially similar solutions, with the aim of avoiding the production of offspring that perform poorly.
- To maintain enough diversity in the population, with the aim of avoiding premature convergence.

In general, the clustering problem is to find q groups of similar elements, from a collection of p elements, where $p > q$ [130].

Clustering techniques are grouped into two different classes: hierarchical and partitional clustering methods.

Hierarchical clustering techniques organize the data into a nested sequence of groups. Partitional clustering methods generate a single partition of the data in an attempt to recover natural groups present in the data [58]. Hierarchical clustering methods, generally require only the proximity matrix among the objects, whereas partitional techniques expect the data in the form of a pattern matrix.

- **Hierarchical Agglomerative Clustering Algorithms**: These methods start with disjoint clustering, which places each of the n objects in an individual cluster. The process concludes with a procedure for transforming a proximity matrix into a sequence of nested partitions. Two algorithms belonging to this class are briefly described next.
  - **Johnson's algorithm:** This algorithm is a hierarchical clustering method [63]. The pseudocode of this algorithm is shown in Algorithm 8.

Algorithm 8 Single-link clustering

1: Begin with the disjoint clustering having level $L(0) = 0$ and sequence number $m = 0$.
2: **repeat**
3: Find the least dissimilar pairs of clusters in the current clustering, say pair $(r), (s)$,
4: According to $d[(r), (s)] = min\{d[(i), (j)]\}$ where the minimum is over all pairs of clusters in the current clustering.
5: Increment the sequence number: $m = m + 1$. Merge clusters (r) and (s) into a single cluster to form the next clustering m. Set the level of this clustering to $L(m) = d[(r), (s)]$
6: Update the proximity matrix, D, by deleting the rows and columns corresponding to clusters (r) and (s) and adding a row and a column corresponding to the newly formed cluster. The proximity between the new cluster, denoted (r, s) and the old cluster (k) is defined as: $d[(k), (r, s)] = min\{d[(k), (r)], d[(k), (s)]\}$
7: **until** objects are not in N clusters.

  - **Average linkage method:** Another well-known hierarchical algorithm is the average linkage method proposed by Morse [90] which has been found successful in practice. The pseudocode of this algorithm is shown in Algorithm 9.

Algorithm 9 Average-linkage method

1: **output**
2: $\bar{P}$ external set
3: $\bar{N}$ external set's maximum size
4: **output**
5: $\bar{P}_{t+1}$ updated external set
6: Initialize cluster set C; each individual $i \in \bar{P}$ constitutes a distinct cluster: $C =_{i \in \bar{P}} \{\{\text{i}\}\}$.
7: **while** $C > \bar{N}$ **do**
8: Calculate the distance of all possible pairs of clusters. The distance d_c of two clusters c_1 and $c_2 \in C$ is given as the average distance between pairs of individuals across the two clusters.
9: Determine two clusters c_1 and c_2 with minimal distance d_c; the chosen clusters amalgamate into a larger cluster: $C = C\{c_1, c_2\}u\{c_1 u c_2\}$
10: **end while**

- **Partitional clustering methods:** The K-means algorithm is a well-known partitional clustering method. Its methodology consists of dividing the data into K clusters where K is a user-specified parameter. The K-means algorithm starts with K centroids (K randomly selected solutions, represent the K centroids) and iteratively performs the following steps [39]:

  - assign each data instance to the cluster whose centroid is nearest to that instance;
  - compute the new centroids of each cluster.

  These two steps are iteratively performed until no data instance moves from one cluster to another.

  The basic idea is to cluster each solution to its nearest centroid, such that the distance between each instance and its nearest centroid is minimized.

3.3.4 Variation Operators

The goal of variation operators is to create new non-dominated individuals derived from the old ones. The variation operators can be used for exploration (search in an untested region of the search space) or exploitation (search for better solutions in the vicinity of good ones). Variation operators should be used with a strong exploration emphasis at the beginning of the search, and with an exploitation emphasis at the end of the execution of the evolutionary algorithm.

Variation operators vary according to the evolutionary paradigm adopted. It is also important to note that variation operators are representation-dependent. That is, for different representations different variation operators have to be defined. The most widely used variation operators are mutation and recombination.

3.3.4.1 Mutation

The mutation operator is used to create a new individual solution from an old one by applying some kind of randomized change to the genotype. Mutation is normally used as an exploration operator.

3.3.4.2 Recombination

The role of recombination is to create new individual solutions from the information contained within two or more parent solutions, with the aim that each new offspring has desirable features from its ancestors, but different to them. Recombination is usually adopted as an exploitation operator.

Since there are many recombination operators proposed in the specialized literature, choosing a "good" recombination operator plays an important role when we try to solve a specific multiobjective optimization problem. Toscano and Coello [114] found that there are recombination operators that perform better than others in multiobjective optimization problems (e.g. simulated binary crossover [26] performs better than arithmetical crossover in the presence of a disjoint Pareto optimal set).

3.3.5 Survivor Selection Mechanism

The survivor selection mechanism[3] is similar to the parent selection mechanism, but it is used at a different stage of the evolutionary cycle. The survivor selection mechanism is invoked after having created the offspring from the parents selected [31]. Since the population size in an evolutionary algorithm is usually constant we need to decide which individuals (from the union of parents and offspring) will be allowed to survive for the next generation. DeJong [30] proposed a methodology that avoids a performance degradation by copying the best individual from the parent generation to the next. This strategy is commonly known as elitism. However, as the notion of Pareto dominance is normally used in multiobjective optimization, then the design of an elitist mechanism becomes elusive. One could retain all the non-dominated individuals generated (since they all are conceptually equally good). However, this is normally impractical because of the very large number of solutions that may be necessary to store.

A possible solution is to use a historical archive where the non-dominated solutions are stored. Historical archives have been studied in more detail in recent years, and a number of mechanisms have been proposed to encourage a good distribution of the solutions stored in the archive [71, 80, 70, 66, 69].

Since it is normally the case that a bound is imposed on the size of the external archive, an additional selection mechanism is usually necessary to discriminate among non-dominated solutions. This additional mechanism should choose those solutions which are better spread along the known Pareto front (It is important to note that the survivor selection mechanism should be applied in objective function space, whereas the parent selection mechanism should be applied in design variable space).

The easiest possible implementation in this regard is a historical archive in which all the non-dominated vectors will be stored after each generation (removing any dominated solutions). The adaptive-grid is another algorithm commonly used [71]. It is an external file with a diversity approach based on geographical distribution of solutions in objective function space. ϵ-dominance proposed in [80] is a fast algorithm for maintaining the diversity based on ϵ-dominance and ϵ-Pareto optimality [87]. Clustering techniques [39] are also suitable to be included in the survivor selection mechanism.

[3] In this work, survivor selection mechanism and replacement mechanism will be used interchangeably.

3.3.5.1 Using a Historical Archive of Solutions

The basic idea is to use an external archive to store all the solutions that are non-dominated with respect to the contents of the archive.

The function of the external archive is to decide whether a certain solution should be added or not to the archive. The decision-making process is the following (the same but in pseudocode form is shown in Algorithm 10).

After the MOEA finishes one cycle, the non-dominated vectors found are extracted from the final population and are compared (on a one-per-one basis) with respect to the contents of the external archive. If the external archive is empty, then the current solution is accepted (see case 1 in Figure 3.2). If this new solution is dominated by an individual within the external archive, then such a solution is automatically discarded (see case 2 in Figure 3.2). Otherwise, if none of the elements contained in the external population dominates the solution wishing to enter, then such a solution is stored in the external archive (see case 3 in Figure 3.2). If there are solutions in the archive that are dominated by the new element, then such solutions are removed from the archive (see case 4 from Figure 3.2).

Finally, the external population is normally bounded in size. If this is the case, once the external population has reached its maximum allowable capacity, an extra mechanism is invoked to encourage diversity in the archive (see case 5 from Figure 3.2). Otherwise, the external archive will not accept any other solution, until a space is freed.

Algorithm 10 External archive(solution)

```
if external-archive E = φ then
    store solution in E
else if solution dominates E then
    eliminate dominated individuals
    store solution in E
else if solution is not dominated by E then
    if E is not full then
        store solution in E
        Call-an-additional-mechanism-to-keep-diversity
    end if
end if
```

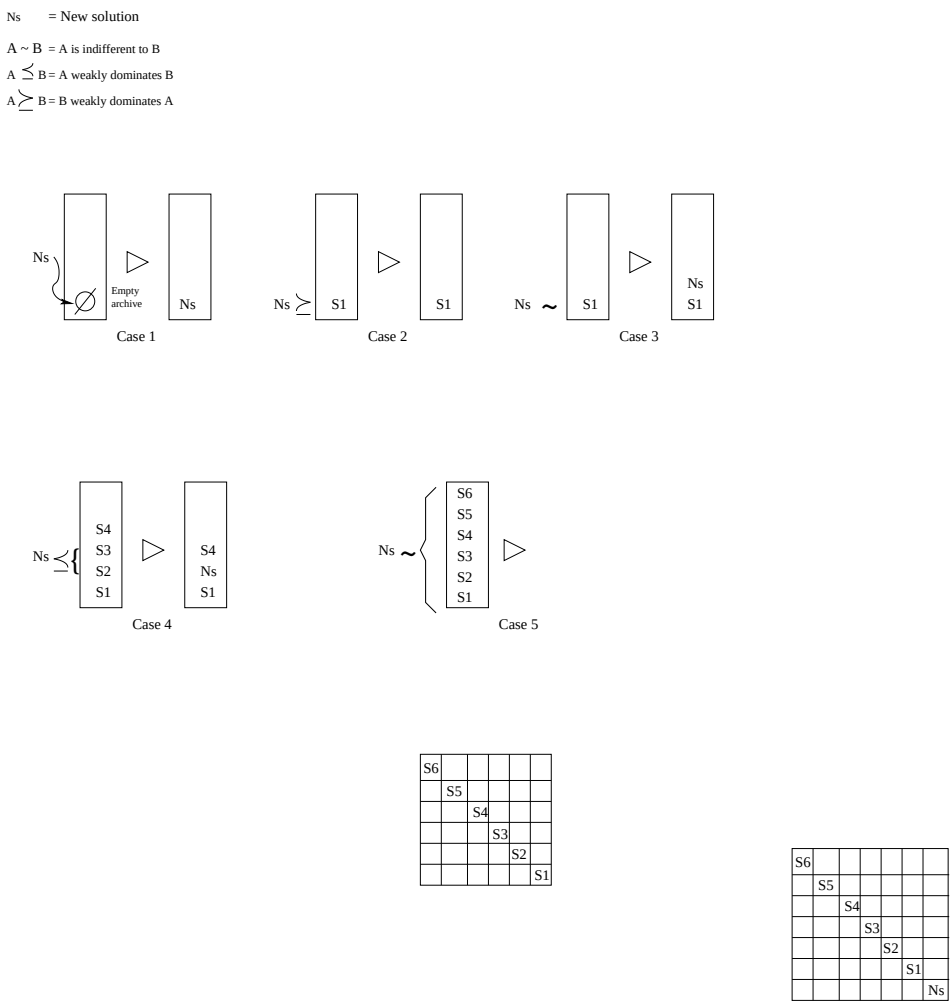

Figure 3.2: Possible decision cases to store for the external archive.

4

Multiobjective Optimization Techniques

4.1 Introduction

SINCE the economist *Vilfredo Pareto* introduced in 1896 the *compromise solution* concept [95], several multiobjective optimization techniques have been developed.

This chapter introduces the state-of-the-art related to multiobjective optimization. The main traditional and evolutionary multiobjective methods in current use are briefly reviewed in Sections 4.2 and 4.3, respectively.

4.2 Traditional Techniques

The Operations Research community has developed more than 30 multiobjective optimization algorithms [92, 87]. A general classification of mathematical programming techniques used for multiobjective optimization is shown in Figure 4.1. The main approaches included in this taxonomy are briefly discussed next.

4.2.1 No Preference Information

We consider within this group simple methods where the decision maker does not provide any information regarding the type of non-dominated solutions that he or she prefers. These methods are usually adopted when the decision maker does not have any particular preference for certain types of solutions and is satisfied with any Pareto optimal solution [87].

Some methods included under this category are:

- Method of the global criterion [125, 127].

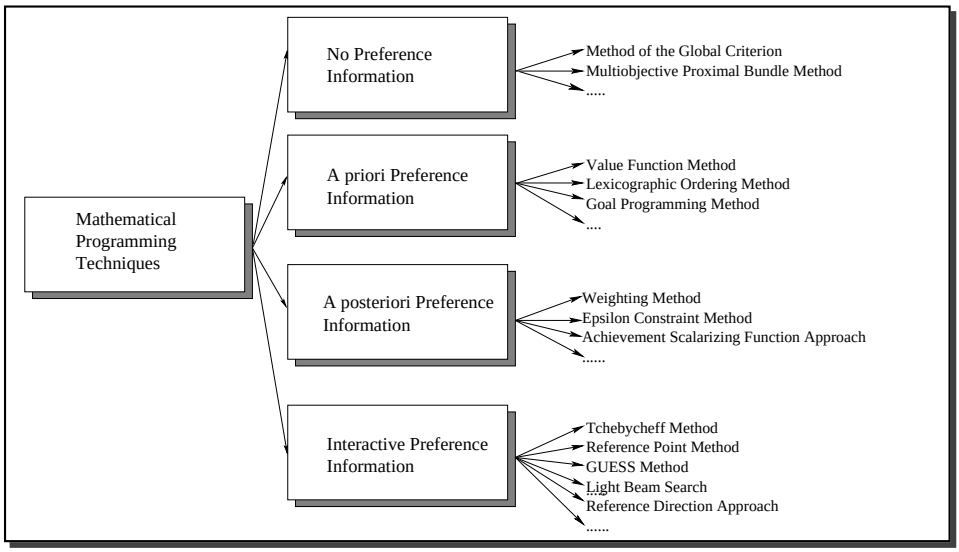

Figure 4.1: A classification of some methods used to conduct multiobjective optimization using mathematical programming techniques

- Multiobjective proximal bundle method [83].

4.2.2 A priori Methods

In these methods the preference specification is provided *before* the solution process. The main drawback of these methods is that the desirable features for the solutions of an arbitrary problem are rarely known in advance. The most important techniques within this class are the following:

- Value function method [64].
- Lexicographic ordering method [33].
- Goal programming [11]

4.2.3 A posteriori Methods

These methods produce a Pareto optimal set (or part of it), through the use of a computationally expensive process. The decision maker selects the most preferred solution from among all the available alternatives.

These methods can be divided into two classes: those that can find the whole Pareto front, and those that can only generate those points which are located on each farthest end of the Pareto front.

The main advantages of these methods are that the solutions are independent from the decision maker's preference. Therefore the analysis has to be performed only once. However, they have some disadvantages such as the computational cost, which in most of these methods tends to be prohibitively high (computationally speaking) and usually there are too many solutions to choose from, which complicates the decision-making process.

Some examples of this type of approach are shown below:

- Weighting method [40, 126].
- ϵ-constraint method [47].
- Method of weighted metrics [127].
- Achievement scalarizing function approach [121].

4.2.4 Interactive Methods

This type of method is more developed than the classes previously described. Methods in this class work based on the hypothesis that the decision maker is unable to indicate preference information *a priori* because of the complexity of the problem. Thus, the decision maker specifies and adjusts his or her preferences at the same time as he or she is learning more about the problem. Andersson [2] lists some advantages of this type of method:

- There is no need for *a priori* preference information.
- Only local preference is needed.
- It is a learning process in which the decision maker gets a better understanding of the problem.
- As the decision maker takes an active part in the search it is more likely that he or she accepts the final solution.

And some disadvantages:

- The solutions depend of how well the decision maker can articulate his or her preferences.
- A high effort is required from the decision maker during the whole search process.
- The solution depends of the behavior of the decision maker.

The basic algorithm for multiobjective optimization used by this type of approach [87] can be seen in Algorithm 11.

Algorithm 11 Basic algorithm of an interactive method

```
Find a feasible solution
repeat
  Interact with the decision maker (introduction of preferences)
  Obtain a new solution according to the user's preferences
until a solution is accepted
```

Some methods grouped under this category are the following:

- Interactive surrogate worth trade-off method [10].

- Geoffrion-Dyer-Feinberg method [42].
- Sequential proxy optimization technique [105].
- Tchebycheff method [112].
- Reference point method [122].
- GUESS method [7].
- Light Beam search [59].
- Reference direction approach [73].
- NIMBUS method [88].

4.2.5 Drawbacks of the Conventional Techniques

Traditional multiobjective methods are usually computationally expensive. Consequently, it is difficult to obtain solutions in polynomial time if we increase the complexity of the problem. And, even when these methods reach the true Pareto front of a problem, they generate only one solution at a time (i.e., they need to be run several times from different starting points). Furthermore, traditional mathematical programming techniques are normally highly susceptible to the shape or continuity of the Pareto front.

4.3 Evolutionary Techniques

The first notion about the use of evolutionary algorithms in multiobjective problems dates back to the mid-1960s. Rosenberg's Ph. D. thesis [102] included a suggestion about how to use multiple properties (closeness with respect to some chemical composition) in his simulation of genetic and the chemistry of single-cell organisms. Since his implementation comprised a single property, he could not show a multiobjective approach in his work. However, this work was the first indicative of the use of evolutionary algorithms to deal with several objective functions.

The first practical implementation of a multiobjective evolutionary algorithm was performed by Schaffer [106] in the mid-1980s. After this novel work, practically there was no interest in the area until the publication of Goldberg's book [43] in which the concept of "Pareto ranking" was introduced. Pareto ranking is a selection scheme which basically consists on selecting individuals based not on their fitness, but on Pareto dominance, in such a way that the non-dominated individuals (i.e. the non-dominated vectors) of the population have all the highest fitness (they all have the same fitness).

In the mid-1990s the area now called "evolutionary multiobjective optimization" had its greatest development. It was during that period that researches proposed algorithms that became widely used (e.g. MOGA [37], the NSGA [110] and the NPGA [55]). The last ten years have produced more sophisticated algorithms which are both more effective and efficient than their predecessors (e.g., the NSGA-II [110], PAES [71], SPEA [137], SPEA2 [134]).

The area has become increasingly popular in the last few years as reflected by a considerable number of publications [1].

There are several mutiobjective evolutionary algorithms in the specialized literature. *Coello* [13] classifies them as follows:

- Simple approaches:
  - Aggregating functions.
  - Goal programming.
  - Goal attainment.
  - ϵ-constraint method.
- Approaches not based on the notion of Pareto optimum.
  - VEGA.
  - Lexicographic ordering.
  - Using gender to identify objectives.
  - Weighted Min/Max approach.
  - Use of randomly generated weights and elitism.
- Approaches based on the notion of Pareto optimum.
  - MOGA.
  - NSGA and NSGA-II.
  - NPGA.
  - SPEA and SPEA2.
  - PAES.
  - MicroGA.

Next, we will provide a brief description of the most significant characteristics of the approaches that are most representative of the area (the others are described in detail somewhere else [13, 19]).

4.3.1 Vector Evaluated Genetic Algorithm (VEGA)

David Schaffer modified [107] the GENESIS' [45] selection process so that it could handle multiple objectives. The resulting algorithm was called the Vector Evaluated Genetic Algorithm (VEGA). This algorithm is classified as a criterion selection technique. The general idea is that, a population of size M must be subdivided into k subpopulations (where k is the number of objectives). These subpopulations are then shuffled together to obtain a new population of size M. Finally crossover and mutation are applied to the resulting population. This process is performed until a stop condition is reached. The pseudocode of this algorithm is shown in Algorithm 12.

[1] See the EMOO repository which contains over 2000 bibliographic references: http://delta.cs.cinvestav.mx/~ccoello/EMOO

Algorithm 12 VEGA Algorithm

```
Initialize population with randomly generated solutions
Evaluate each solution
while stop condition is not reached do
  Generate k sub-populations
  Perform proportional selection to each subpopulation
  Mix the individuals from the subpopulations into one population
  Apply crossover
  Apply mutation
  Evaluate new candidates
end while
```

4.3.2 Multiobjective Genetic Algorithm (MOGA)

This approach was proposed by Fonseca & Fleming [37]. In this technique, the ranking process is based on Pareto dominance (i.e. the non-dominated individuals will be assigned rank 1, and the ranking assigned to the other individuals proportionally increases with respect to the number of individuals that dominate them). The rank position is then given by the following expression:

$$Rank(x_i, t) = 1 + p_i^t \tag{4.1}$$

where:

x_i = the ith individual.

t = current generation.

p_i^t = number of individuals that dominate to the ith individual at generation t .

Fitness assignment is performed in the following way [37]:

1. Sort population according to rank.
2. Assign fitness to individuals by interpolating from the best (rank 1) to the worst (rank $n \leq M$) in the way proposed in [43], according to some function, usually linear, but not necessarily.
3. Average the fitness of individuals with the same rank, so that all of them are sampled at the same rate. This procedure keeps the global population fitness constant while maintaining appropriate selective pressure, as defined by the function used.

The pseudocode of MOGA is shown in Algorithm 13.

4.3.3 Niched Pareto Genetic Algorithm (NPGA)

This approach, proposed by Horn & Nafpliotis [54], applies a Pareto dominance based tournament selection on a subset of the whole population. The methodology consists on

Algorithm 13 MOGA Algorithm

```
Initialize population with randomly generated solutions
Evaluate each solution
while stop condition is not reached do
    Apply rank based on Pareto dominance
    Compute niche count
    Apply linearly scaled fitness
    Apply shared fitness
    Apply selection via stochastic universal sampling
    Apply single point crossover
    Apply mutation
    Evaluate new candidates
end while
```

comparing two individuals with respect to a (randomly chosen) subset of the population (tournament size is the parameter that indicates the size of the subset). If after this comparison, there is no single winner (i.e. there is a single winner only if one of the contenders is non-dominated with respect to the subset adopted and the other is dominated.), then *fitness sharing* is used to solve the tie [44, 54]. The performance of the approach may be negatively affected if a wrong tournament size value is selected (e.g., if a very small size is adopted). The pseudocode of the NPGA is shown in Algorithm 14.

Algorithm 14 NPGA Algorithm

```
Initialize population with randomly generated solutions
Evaluate each solution
while stop condition is not reached do
    Apply specialized binary tournament selection
    if both candidates are dominated or non-dominated then
        Apply specialized fitness sharing:
        Select candidate with lower niche count
    end if
    Apply single point crossover
    Apply mutation
    Evaluate new candidates
end while
```

4.3.4 Non-dominated Sorting Genetic Algorithm (NSGA)

The Non-dominated Sorting Genetic Algorithm was proposed by Srinivas & Deb in 1994 [110]. The NSGA is based on several layers of classifications of the individuals as suggested by Goldberg [43]. Before selection is performed, the population is ranked on the basis of non-domination: all non-dominated individuals are classified into one category (with a dummy fitness value, which is proportional to the population size, in order to provide an equal reproductive potential for these individuals). Then, this group of classified

individuals is ignored and another layer of non-dominated individuals is considered. The process continues until all individuals in the population are classified. The NSGA uses fitness sharing to maintain diversity.

Since individuals in the first front have the maximum fitness value, they always get more copies than the rest of the population. This allows to search for non-dominated regions, and results in convergence of the population toward such regions. Figure 4.2 shows the use of the layers in the NSGA, and Algorithm 15 shows the NSGA's pseudocode.

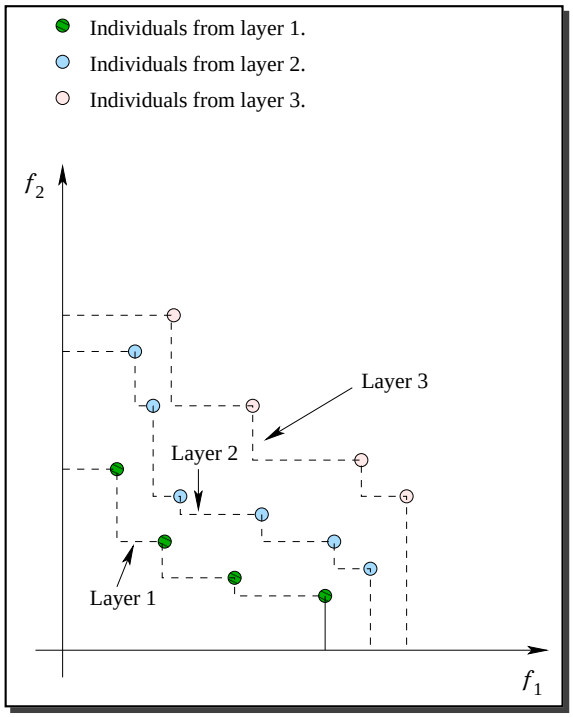

Figure 4.2: Graphical representation of the layers adopted by the NSGA.

A second version of this algorithm, called *NSGA-II*, was recently proposed by *Deb et al.* [28]. This algorithm solves most of the problems of the original version [13].

The NSGA-II is more efficient (computationally speaking) than the original NSGA [110], it uses elitism and a crowded comparison operator that keeps diversity without specifying any additional parameters. It also incorporates elitism (through the use of $(\mu+\lambda)$-selection), a crowded comparison operator and it keeps diversity without specifying any additional parameters. It remains as one of the most competitive multiobjective evolutionary algorithms known to date.

4.3.5 Strength Pareto Evolutionary Algorithm (SPEA)

The Strength Pareto Evolutionary Algorithm (SPEA) was proposed by Zitzler & Thiele [137]. SPEA uses a mixture of preestablished techniques [136]. However, this algorithm was a novelty, since it introduced the use of an external non-dominated set (which is an archive containing non-dominated solutions previously found) to keep a historical record of the non-dominated vectors found along the evolutionary process. To manage diversity

Algorithm 15 NSGA Algorithm

```
Initialize population with randomly generated solutions
Evaluate each solution
while stop condition is not reached do
  Apply rank based on Pareto dominance in each layer
  Compute niche count
  Apply shared fitness
  Apply selection via stochastic universal sampling
  Apply single point crossover
  Apply mutation
  Evaluate new candidates
end while
```

on this external set, Zitzler & Thiele used a clustering technique called "average linkage method" [90].

The pseudocode of SPEA is shown in Algorithm 16.

A second version of this algorithm called *SPEA2* was recently proposed by *Zitzler et al.* [134]. The main differences of SPEA2 in comparison to SPEA are [134, 133]:

- An improved fitness assignment scheme is used, which takes into account for each individual how many individuals it dominates and how many dominate it.
- A nearest neighbor density estimation technique is incorporated which allows a more precise guidance of the search process.
- A new archive truncation methods guarantees the preservation of boundary solutions.

Algorithm 16 SPEA Algorithm

```
Initialize population P with randomly generated solutions
Evaluate each solution
Create empty external set E
while stop condition is not reached do
  Copy non-dominated members of P to E
  Remove elements from E which are covered by any other member of E
  if maximum capacity of E has been exceeded then
    Decide the elements to remove using clustering (and delete them)
  end if
  Evaluate individuals from P and E
  Apply binary tournament selection to select individuals from P + E
  Apply crossover
  Apply mutation
end while
```

4.3.6 Pareto Archived Evolution Strategy (PAES)

This algorithm (whose pseudocode is shown in Algorithm 17) was introduced by Knowles & Corne [71]. PAES consists of a (1+1) evolution strategy (i.e., a single parent that generates a single offspring) in combination with a historical archive that records some of the non-dominated solutions previously found. This archive is used as a reference set against which each mutated individual is compared. Such a historical archive is the elitist mechanism adopted in PAES. However, an interesting aspect of this algorithm is the procedure used to maintain diversity which consists of a crowding procedure that divides objective function space in a recursive manner. Each solution is placed in a certain grid location based on the values of its objectives (which are used as its "coordinates" or "geographical location"). A map of such grid is maintained, indicating the number of solutions that reside on each grid location. Since the procedure is adaptive, no extra parameters are required (except for the number of divisions of the objective space).

Algorithm 17 PAES Algorithm

```
Initialize single population parent p
Evaluate p
Add p to the archive
while stop condition is not reached do
  Mutate p to produce child c and evaluate fitness
  if p dominates c then
    Discard c
  else if dominates p then
    Replace p with c, and add c to the archive
  else
    Determine if p or c will be the new current solution and
    Determine whether to add c to the archive
  end if
end while
```

4.3.7 Multiobjective Micro Genetic Algorithm (MicroGA)

This approach was introduced by Coello Coello & Toscano Pulido [16, 17]. A microgenetic algorithm is a GA with a small population size (≤ 5 individuals) and a reinitialization process. The way in which the microGA works is illustrated in the pseudocode shown in Algorithm 18. First, a random population is generated. This random population feeds the population memory, which is divided in two parts: a replaceable and a non-replaceable portion. The non-replaceable portion of the population memory never changes during the entire run and is meant to provide the required diversity for the algorithm. In contrast, the replaceable portion experiences changes after each cycle of the microGA.

The population of the microGA at the beginning of each of its cycles is taken (with a certain probability) from both portions of the population memory so that there is a mixture of randomly generated individuals (non-replaceable portion) and evolved individuals (replaceable portion). During each cycle, the microGA undergoes conventional genetic operators.

After the microGA finishes one cycle, two non-dominated vectors are chosen[2] from the final population and they are compared with the contents of the external memory (this memory is initially empty). If either of them (or both) remains as non-dominated after comparing it against the vectors in this external memory, then it is included there (i.e., in the external memory). This is the historical archive of non-dominated vectors. All dominated vectors contained in the external memory are eliminated.

The microGA uses then three forms of elitism: (1) it retains non-dominated solutions found within the internal cycle of the microGA, (2) it uses a replaceable memory whose contents is partially "refreshed" at certain intervals, and (3) it replaces the population of the microGA by the nominal solutions produced (i.e., the best solutions found after a full internal cycle of the microGA).

Algorithm 18 MICROGA Algorithm

```
Generate randomly the initial population P of size N and store P in the population
memory M_r and M_e
i=0
while i < GMax do
  Get P^t from M
  repeat
    Apply binary tournament selection
    Apply two-point crossover
    Apply uniform mutation
    Apply elitism
    Produce a new population
  until nominal convergence is reached
  Copy solutions to external memory
  if i mod replacement cycle then
    Apply second form of elitism
  end if
  i=i+1
end while
```

4.3.8 ϵ-MOEA

ϵ-Multiobjective Evolutionary Algorithm (ϵ-MOEA) was proposed by Deb *et al* [27]. ϵ-MOEA is based on the ϵ-dominance concept introduced in [79]. The core idea [27] is to have two population co-evolving: the EA population P and the archive population E. The procedure begins initializing the population P with random solutions. Then P feeds E with ϵ-non-dominated solutions. Next, two solutions (p and e) are randomly selected, one from P and one from E. After that, p and e are mated to create λ offspring solutions. Finally, each offspring solution is compared with P and E for their possible inclusion. The mechanism to maintain diversity lies consists of dividing the search space into a certain number of hyper-

[2]This is assuming that there are two or more non-dominated vectors. If there is only one, then this vector is the only one selected.

boxes. Such hyper-boxes will maintain the algorithm's diversity, since each hyper-box can be occupied by only a single solution.

A graphical illustration [27] of the ϵ-MOEA procedure is shown in Figure 4.3 while its pseudocode is shown in Algorithm 19.

Algorithm 19 ϵ-MOEA Algorithm

```
Generate randomly the initial population P(0) of size N
Initialize E(0) from P(0)'s ε-non-dominated solutions
i=0
while i < GMax do
  Select the solution p from P(i)
  Select the solution e from E(i)
  Apply mating to select λ offspring
  P ← P ∪ λ | λ ε-dominates P
  E ← E ∪ λ | λ ε-dominates E
  i=i+1
end while
```

4.3.9 Multiobjective Particle Swarm Optimization

There have been several recent proposals to extend PSO to handle multiple objectives. We will review next the most important of them:

- **The algorithm of Moore and Chapman [89]**: This algorithm was presented in an unpublished document and it is based on Pareto dominance. The authors emphasize the importance of performing both an individual and a group search (a cognitive component and a social component). However, the authors did not adopt any scheme to maintain diversity.

- **The Swarm Metaphor of Ray & Liew [100]**: This algorithm uses Pareto dominance and combines concepts of evolutionary techniques with the particle swarm. The approach uses crowding to maintain diversity and a multilevel sieve to handle constraints (for this, the authors adopt the constraint and objective matrices proposed in some of their previous research [99]).

- **The algorithm of Parsopoulos & Vrahatis [96]**: This algorithm adopts an aggregating function (three types of approaches were implemented: a conventional linear aggregating function, a dynamic aggregating function and the bang-bang weighted aggregation approach [62] in which the weights are varied in such a way that concave portions of the Pareto front can be generated).

- **Dynamic Neighborhood PSO proposed of Hu and Eberhart [56]**: In this algorithm, only one objective is optimized at a time using a scheme similar to lexicographic ordering. Lexicographic ordering tends to be useful only when few objective

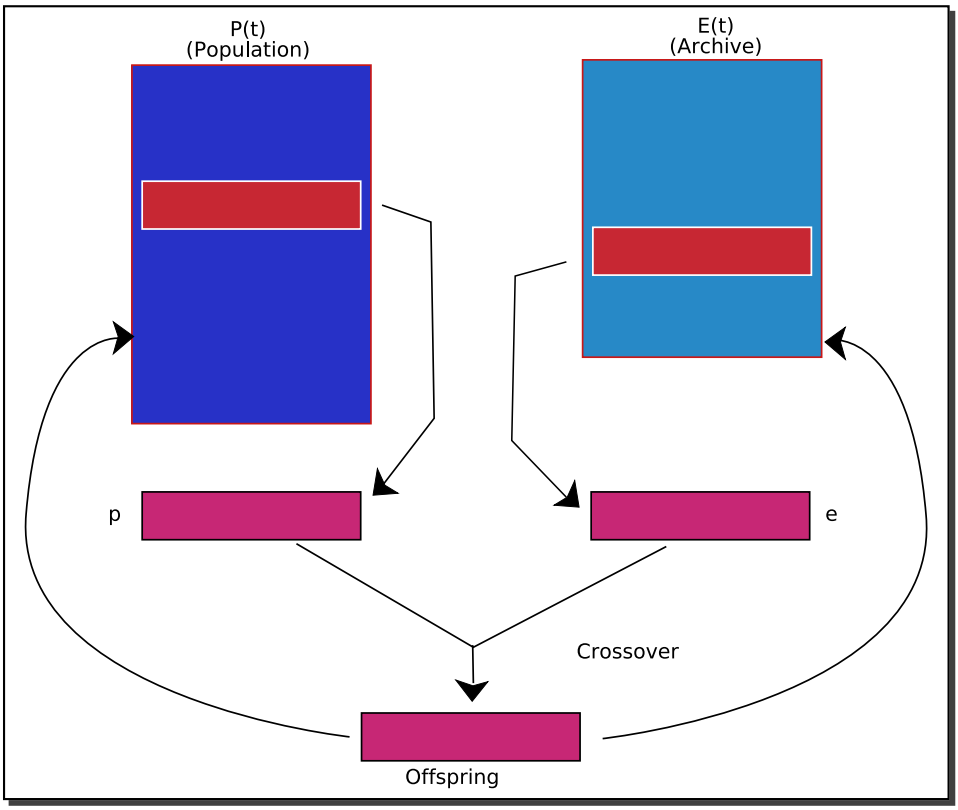

Figure 4.3: Illustration of the ϵ-MOEA procedure

functions are used (two or three), and it may be sensitive to the ordering of the objectives. The idea of the dynamic neighborhood is, however, quite interesting and is novel in this context.

- **The Multiobjective Particle Swarm Optimizer (MOPSO) of Coello & Lechuga [15]**: This proposal is based on the idea of having a global repository in which every particle will deposit its flight experiences after each flight cycle. Additionally, the updates to the repository are performed considering a geographically-based system defined in terms of the objective function values of each individual; this repository is used by the particles to identify a leader that will guide the search.

  This approach also uses a mutation operator that acts both on the particles of the swarm and on the range of each design variable of the problem to be solved. The pseudocode of MOPSO is shown in Algorithm 20.

- **The approach of Fieldsend & Singh [32]**: This approach incorporates an unconstrained elite archive (in which a special data structure called "dominated tree" is adopted) to store the non-dominated individuals found along the search process. The archive interacts with the primary population in order to define local guides. This approach also uses a "turbulence" operator which is basically a mutation operator that acts on the velocity value used by PSO.

- **The algorithm of Mostaghim & Teich [91]:** This approach uses a sigma method in which the best local guides for each particle are adopted to improve the convergence and diversity of a PSO algorithm used for multiobjective optimization. They also use a "turbulence" operator, but applied on decision variable space. The use of the sigma values increases the selection pressure of PSO (which was already high). This may cause premature convergence in some cases (e.g., in multifrontal problems). In this sort of approach, the authors provide comparisons with SPEA2 [134] and the dominated trees of Fieldsend & Singh [32] using four test functions and the coverage metric.

- **The modified Dynamic Neighborhood of Hu et al. [57]**: This approach uses a secondary population (called "extended memory") and introduces some further improvements to their dynamic neighborhood PSO approach [56]. Nevertheless, it is worth indicating that this approach completely fails in generating the true Pareto front of some problems (see [57] for details). Hu et al. [57] also compared their algorithm with respect to the Strength Pareto Evolutionary Algorithm (SPEA) [137] using the set coverage metric [130].

- **The non-dominated Sorting PSO of Li [81]**: This approach incorporates the main mechanisms of the NSGA-II [28] into a PSO algorithm. The proposed approach showed a very competitive performance with respect to the NSGA-II (even outperforming it in some cases).

- **The PS-EA of Srinivas and Hou [111]**: The Particle Swarm Inspired Evolutionary Algorithm (PS-EA), is a hybrid between PSO and an evolutionary algorithm. The authors argue that the traditional PSO equations are too restrictive when applied to multi-constrained search spaces. Thus, they propose to replace the PSO equations with the so-called self-updating mechanism, which emulates the workings of the equations. Such mechanism uses an inheritance probability tree to update each individual in the population. An interesting aspect of this approach is that the authors also use a mechanism to dynamically adjust the inheritance probabilities in the inheritance probability tree based on the status of the algorithm at a certain moment in time. The approach uses a memory to store the elite particles and does not use a recombination operator.

- **The approach of Zhang et al. [128]**: This approach attempts to improve the selection of g_{best} and p_{best} when the velocity of each particle is updated. For each objective function, there exist both a g_{best} and a p_{best} for each particle. In order to update the velocity of a particle, the algorithm defines the g_{best} of a particle as the average of the complete set of g_{best} particles. Analogously, the p_{best} is computed using either a random choice or the average from the complete set of p_{best} values. This choice depends on the dispersion degree between the g_{best} and p_{best} values of each particle.

- **The approach of Bartz-Beielstein et al. [4]**: This approach starts from the idea of introducing elitism (archiving) into PSO. Different methods for selecting and deleting particles from the archive are analyzed to generate a satisfactory approximation of the

Pareto front. The selection methods analyzed are based on the contribution of each particle to the diversity of the Pareto front. Deleting methods are either inversely related to the selection fitness or based on the previous success of each particle. The authors provide some statistical analysis in order to assess the impact of each of the parameters used by their approach.

- **The approach of Baumgartner et al. [5]**: This approach uses weighted sums (i.e., linear aggregating functions) to solve multiobjective optimization problem. In this approach, the swarm is equally partitioned into n subswarms, each of which uses a different set of weights and evolves into the direction of its own swarm leader. The approach adopts a gradient technique to identify the Pareto optimal solutions.

- **The Multi-Species PSO of Chow and Tsui [67]**: These authors proposed an autonomous agent response learning algorithm. They propose to decompose the award function into a set of local award functions and, in this way, to model the response extraction process as a multiobjective optimization problem. A modified PSO called "Multi-Species PSO" is introduced by considering each objective function as a species swarm. A communication channel is established between the neighboring swarms for transmitting the information of the best particles, in order to provide guidance for improving their objective values. Also, the authors propose to modify the equation used to update the velocity of each particle, considering also the global best particle of its neighboring species.

- **The AWPSO of Mahfouf et al. [82]**: This is an enhancement of the original PSO algorithm which aims to improve the performance of this heuristic in multiobjective optimization problems. The approach is called the Adaptive Weighted PSO (AWPSO) algorithm, and its main idea is to modify the velocity by including an acceleration term which increases with the number of iterations. This aims to enhance the global search ability of the algorithm towards the end of the run thus helping the approach to escape from local optima. A weighted aggregating function is also used to guide the selection of the personal and global best leaders. The authors use dynamic weights to generate different elements of the Pareto optimal set. A non-dominated sorting scheme is adopted to select the particles from one iteration to the next one. The approach was applied to the design of heat treated alloy steels based on data-driven neural-fuzzy predictive models.

There are a few things worth saying about these previous proposals. Some of them are not based on Pareto dominance (e.g. [5, 96, 56, 82]). Others are really hybrid approaches and deviate in an important way from the main precepts of the PSO algorithm (e.g. [67, 111, 100]). The remaining approaches adopt mechanisms that are more standard in evolutionary multiobjective optimization (i.e., Pareto-based selection and elitism). However, some of them haven't been properly validated (e.g. [89, 128]), and most of them are not available in the public-domain (except for our own MOPSO [15]), and the approach described in [91][3] against which we will compare the performance of our algorithm.

[3]This approach is not really in the public-domain, but we were able to obtain it from her author.

Algorithm 20 MOPSO Algorithm

```
for each particle do
    Initialize x⃗ randomly
    Initialize velocity
    Evaluate fitness function(x⃗)
    Initialize pbest=x⃗
end for
repeat
    for each particle do
        Compute its speed using the following expression on each dimension:
        velocity = W × velocity + random(0,1) × (P_best − x) + random(0,1) × (G_best − x)
        Compute the new position using: x = x + velocity on each dimension
        Evaluate the new position
    end for
    Update the contents of the repository REP
    if x⃗ dominates to P_best then
        P_best = x⃗
    end if
until maximum number of iterations is not reached
```

So, our main motivations for developing a new multiobjective particle swarm optimization were the following:

- We had previously proposed a multiobjective particle swarm optimization algorithm which was highly competitive and that remains as the only one (to date) published in an specialized journal [18]. The experience gained during the development of this approach led us to foresee the high potential of PSO to serve as a basis for more powerful multiobjective optimizers.

- None of the existing approaches was designed with an emphasis on efficiency. Thus, one of our design goals was to perform less than $5,000$ fitness function evaluations, which is considerably below the number of evaluations reported by any of the other multiobjective particle swarm optimizers in current use. In fact, as we will see later on, we were able to design an approach that remains competitive while performing only $2,000$ fitness function evaluations, which is the lowest number of evaluations reported to date for any multiobjective particle swarm optimizer.

- It is a well-known fact within the PSO research community that most multiobjective particle swarm optimizers have problems to converge in some test functions (e.g. [57, 91]), mainly due to a loss of diversity. Additionally, we realized that most researchers in this area had underestimated the importance of the selection of leaders and the intrinsic limitations imposed by the lack of a crossover operator in PSO. We decided to emphasize these issues in our algorithmic design, so that our approach didn't have any of these limitations. In fact, as we will see later on, we introduced on approach based on the use of subsuwarms to generate a good spread of solutions. This approach is novel in the area, an is one of the key contributions of our proposed algorithm.

- Finally, we realized that there were no studies available on the parameters fine-tuning of a multiobjective particle swarm optimizer. Our previous experience in this area (see [18]) indicated us that the studies done for single-objective particle swarm optimizers didn't apply for the multiobjective case. So another goal of our research was to provide a comprehensive (empirical) analysis of the impact of the parameters in the performance of our algorithm in an attempt to fill the gap existing in this regard.

4.3.10 Current Trends

The ranking process is one of the main drawbacks of some of the algorithms previously described, mainly because of its computational cost (this process is $O(kM^2)$ per generation, where k is the number of objective functions and M is the population size). Additionally, an extra mechanism is required to preserve the population's diversity (fitness sharing is usually adopted, which requires a process $O(M^2)$ per generation). However, some recent research has provided new paths to improve the efficiency of evolutionary multiobjective optimization techniques.

Some researchers have focused on (a) decreasing the non-dominance checking and on (b) the development of efficient mechanisms for keeping diversity.

Regarding to the first point (a), the main emphasis has been placed on using more efficient Pareto ranking schemes [61] as well as designing algorithms in which computational efficiency is obtained at the expense of a higher memory usage [60]. Regarding the second point (b), the main emphasis has been on the use of clustering techniques or geographically-based algorithms [130, 131, 71] (i.e. an adaptive grid).

Some researchers have also suggested a distributed genetic algorithm in which Pareto dominance is only applied to the neighbors in a defined neighborhood [103]. This approach can solve simultaneously the problems described above. In this approach, Pareto dominance must be applied to small individual groups in parallel, and there is no need for an additional mechanism to keep diversity, because it emerges naturally from the distributed population. However, to take advantage of these features a parallel architecture is needed.

4.3.11 Advantages and Disadvantages of Evolutionary Algorithms for MOPs

Evolutionary algorithms are particularly suitable for multiobjective optimization mainly because of their flexibility, adaptability and performance. These characteristics make them suitable to solve problems whose Pareto fronts have different shapes (disconnected, convex or concave), without any significant distinction in terms of performance. These characteristics also allow them to locate and exploit promising zones in problems with large search spaces (traditional techniques assume that the Pareto front is convex and that the objective functions are differentiable and continuous). Additionally, multiobjective evolutionary algorithms have the inherent ability of finding several members of the Pareto optimal set in a single run, mainly because they are population-based techniques. Among the evolutionary algorithms' drawbacks are: 1) they can not guarantee neither feasible nor optimality solutions, 2) as a consequence of the No Free Lunch Theorem [124], it is impossible to select *a priori* the best performer algorithm for a selected problem, and 3) similar to last point, it is impossible to select the best performer parameters' values of an algorithm for a predefined

problem.

5

Multiobjective Particle Swarm Optimization

Introduction

PARTICLE swarm optimization (PSO) is a relatively recent heuristic inspired by the choreography of a bird flock which has been found to be quite successful in a wide variety of optimization tasks [65].

Its high speed of convergence and its relative simplicity make PSO a highly viable candidate to be used for solving not only problems with a single objective function, but also multiobjective optimization problems [19]. However, there are important issues that have to be dealt with when extending PSO for solving multiobjective optimization problems (e.g. the selection of leaders).

This chapter is organized as follows: First, a simple modification to the PSO algorithm to be able to handle multiple objectives is presented. Next, the approach is modified to generate multiple solutions in a single run. Then, in order to reduce the number of solutions and increase the distribution of the solutions obtained we add an adaptive grid, the ϵ-dominance concept, and a hyper-plane distribution. Next, the algorithm is modified to maximize the spread of the solutions along the true Pareto front. As a final result, we present an efficient[1] MOEA, whose performance is compared with respect to approaches representative of the state-of-the-art in the area, using standard metrics and test functions, reported in the specialized literature.

[1] where "efficiency" refers to keeping a low number of fitness function evaluations without decreasing performance.

5.1 Multiobjective Particle Swarm Optimization

PSO seems particularly suitable for multiobjective optimization mainly because of the high speed of convergence that the algorithm presents for single-objective optimization [65]. Based on such behavior, one would expect a multiobjective PSO (MOPSO) to be very efficient computationally speaking. However, there is no standard (unique) version of a MOPSO algorithm that had been adopted in the specialized literature.

Next, we present a new proposal developed from scratch to extend the PSO algorithm to deal with several objectives. In order to propose a competitive algorithm, the entire design was directed by the MOP's goals. Based on such goals, three main modifications were performed to the original PSO algorithm.

1. Modify the algorithm to handle multiple objectives and produce a set of non-dominated solutions in a single run.

2. Modify the algorithm to obtain a good distribution of solutions.

3. Modify the algorithm to maximize the spread of the non-dominated solutions.

5.2 Handling multiple objectives

Replacing the comparison operator (to determine whether a solution a is better than a solution b) is a natural modification to a PSO algorithm aimed handle multiple objectives.

The analogy of particle swarm optimization with evolutionary algorithms makes evident the notion that using a Pareto ranking scheme [43] could be the straightforward way to extend the approach to handle multiobjective optimization problems. However, if we merge a Pareto ranking scheme with the PSO algorithm a set of non-dominated solutions will be produced (by definition, all non-dominated solutions are equally good). Having several non-dominated solutions implies the inclusion into the algorithm of both: an additional criteria to decide whether a new non-dominated solution is *pbest* or *gbest* and a strategy to select the guide particles (*pbest* and *gbest*).

In order to manage properly the inclusion of a Pareto ranking scheme into the PSO algorithm, the modifications shown below were performed:

$pbest \leftarrow new\vec{w}_{pos}\ iff\ \ new\vec{w}_{pos} \preceq pbest$. If the current particle's position is non-dominated with respect to its *pbest* position, then the current value of the *pbest* position is replaced with the current solution.

$GBEST \leftarrow new_{pos} \cup GBEST\ iff\ \neg\exists\ \vec{y}^* \in GBEST \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(new_{pos})$. A new position belongs to the *gbest* set if there is no solution in the GBEST set that dominates it.

However, by applying these changes, the selection of an "appropriate" leader becomes a difficult problem, since there can be more than one leader in the GBEST set. Therefore, an additional strategy to select one of the multiple *gbest* to use in the PSO's velocity formula is still necessary. Some possible leader selection strategies for this sake are the following: (1) randomly (a leader is randomly selected—no constraints are imposed on what sort of leader can a particle choose—), (2) the closest (a particle picks as a leader to the geographically

closest leader), and (3) one at a time (in this case a single leader is selected by all the particles at each generation). The third selection strategy has two variants, (3a) particles follow the leader selected unless a new particle's position dominates the current leader, then the new particle becomes the leader to be followed. And (3b), particles follow the leader selected unless they discover a new non-dominated position.

The four schemes were implemented into the PSO algorithm and executed 30 times each in 8 test functions. The outputs of the algorithms are presented in Appendix A. Furthermore, six statistics were measured at each generation.

An analysis from the approaches' results and statistics by test function (See Section 2.5) is performed next:

ZDT1's test function : In Figure A.1 we can see that scheme 2 was the only approach which could reach the true Pareto front. However, scheme 1 was the approach which performed best since the solutions discovered were in a narrower range. Figure A.2 shows the algorithms' behavior. The approach that found more non-dominated solutions (on average) was scheme 2. This algorithm also improved its performance more frequently than the others.

ZDT2's test function : In this test function, three algorithms (see Figure A.3) could reach at least once the true Pareto front (schemes 1, 2 and 3b). However, scheme 1 placed solutions closer to the true Pareto front than the others. This scheme was trapped in a false Pareto front in only two occasions, while the other algorithms were attracted by a false Pareto front more frequently.

Figure A.4 shows that a *pbest* position was updated in almost 1 out of 2 particles in each generation. It also shows that scheme 1 was the algorithm that converged faster and could store more solutions.

ZDT3's test function : From Figure A.6, it is easy to see that the algorithm 2 could not converge. It seems that if it had had more generations to iterate, it would had improved its results more than the others, since the other algorithms seem to have already converged. In Figure A.5, we can see that scheme 1 was the algorithm that performed best.

ZDT6's test function : In this test function the four algorithms could reach the true Pareto front (see Figure A.7). However, in some executions, the algorithms converged to a single solution.

Kursawe's test function : Figure A.9 shows that scheme 2 was the algorithm which performed best, since it was the algorithm that reached more frequently the true Pareto front, and it stored most of its solutions in a narrower range than the others. In Figure A.10, we can see that Algorithm 2 had the best performance (it placed more non-dominated solutions).

Deb1's test function : In Figure A.11 we can see that all the algorithms had no problems to reach the true Pareto front of this test function. Figure A.12 shows that the four algorithms had similar performance. This figure also shows that they converged to the true Pareto front at an early stage of the search.

Deb2's test function : The four algorithms tested placed only a few solutions on the true Pareto front (see Figure A.13). This occurs, mainly because the algorithms converged faster and were trapped in a false Pareto front. Figure A.14 shows that scheme 2 was the algorithm that performed best, since it converged closer to the true Pareto front and stored more non-dominated solutions.

In general, the approaches that performed best were schemes 1 and 2. On the one hand, it is evident that the fact of performing a random selection of a gbest (scheme 1) improves the diversity of the particles and therefore finds more solutions in a single execution. On the other hand, when solutions follow to the closest leader, they improve their position more frequently (they do not spend time flying from one far side of the search space to another).

Since schemes 1 and 2 were the approaches that performed best, we propose to use the union of both (i.e. an algorithm that uses a flip to decide whether to select a random leader or the closest particle as its *gbest*). Results from this algorithm are shown in Appendix A. In Figure A.15, we can see that this algorithm performed best than any of the two separate schemes that originated it. Algorithm 21 shows the pseudo-code of this proposal.

Algorithm 21 MOPSO - Step 1

Require: $nparticles, dimensions, W, C1$ and $C2$

$\vec{gbest} \leftarrow \vec{x_0}$
$GBEST \leftarrow \emptyset$
for $i = 0$ to $nparticles$ **do**
  $\vec{pbest_i} \leftarrow \vec{x_i} \leftarrow initialize_randomly()$
  $fitness_i \leftarrow f(\vec{x_i})$
  $velocity_{id} \leftarrow 0$
  if $\neg\exists\ \vec{y}^* \in GBEST \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness_i})$ **then**
    $GBEST \leftarrow GBEST \cup \vec{x_i}$
  end if
end for
repeat
  for $i = 0$ to $nparticles$ **do**
    if flip(0.5) **then**
      $gbest \leftarrow GBEST_{U(0,|GBEST|)}$
    else
      $gbest \leftarrow y | y \in GBEST \cup \nexists\ z \in GBEST | z - x_i < y - x_i$
    end if
    for $d = 0$ to $ndimensions$ **do**
      $velocity_{id} \leftarrow W \times velocity_{id} + C_1 \times U(0,1) \times (pbest_{id} - x_{id}) + C_2 \times U(0,1) \times (gbest - x_{id})$
      $x_{id} \leftarrow x_{id} + velocity_{id}$
    end for
    $fitness_i \leftarrow f(\vec{x_i})$
    if $fitness_i$ is $\sim$ to $f(\vec{pbest_i})$ **then**
      $\vec{pbest_i} \leftarrow \vec{x_i}$
    end if
    if $\neg\exists\ \vec{y}^* \in GBEST \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness_i})$ **then**
      $GBEST \leftarrow GBEST \cup \vec{x_i}$
    end if
  end for
until Termination criterion

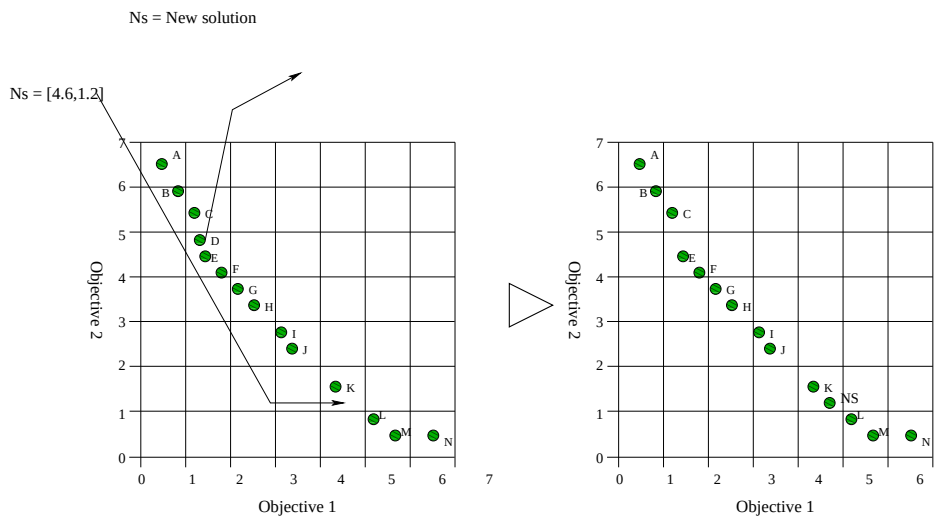

Figure 5.1: Graphical representation of the insertion of a new element in the adaptive grid when the individual lies within the current boundaries of the grid.

5.3 Improving the distribution

The previous Section presented a multiobjective particle swarm optimization which is able to produce several non-dominated solutions in a single execution. However, in some occasions this algorithm finds an excessive quantity of non-dominated solutions (since it does not impose a bound on the total number of solutions). One of the main purposes of any approach that can manage (optimize) multiple objectives is to simplify the work of the decision maker. Therefore, it is necessary to deliver to the decision maker a well distributed set of non-dominated vectors found along the search process.

In order to reduce the non-dominated set, researchers have traditionally used a historical archive to store the non-dominated solutions found during the evolutionary process (see Chapter 3). Such archive needs to have an additional algorithm to decide whether a solution is accepted or not. In this thesis, two main approaches to maintain distribution are implemented, evaluated and discussed: Adaptive Grid and ϵ- dominance. Additionally, a novel methodology to maintain a non-dominated set is proposed.

5.3.1 Adaptive Grid

The basic idea is to use an external archive to store all the solutions that are non-dominated with respect to the contents of the archive. Into the archive, objective function space is divided into regions as shown in Figure 5.1. Note that if the individual inserted into the external population lies outside the current bounds of the grid, then the grid has to be recalculated and each individual within it has to be relocated (see Algorithm 5.2).

The adaptive grid is really a space formed by hypercubes. Such hypercubes have as many components as objective functions has the problem to be solved. Each hypercube can be interpreted as a geographical region that contains an n number of individuals. The main advantage of the adaptive grid is that its computational cost is lower than niching

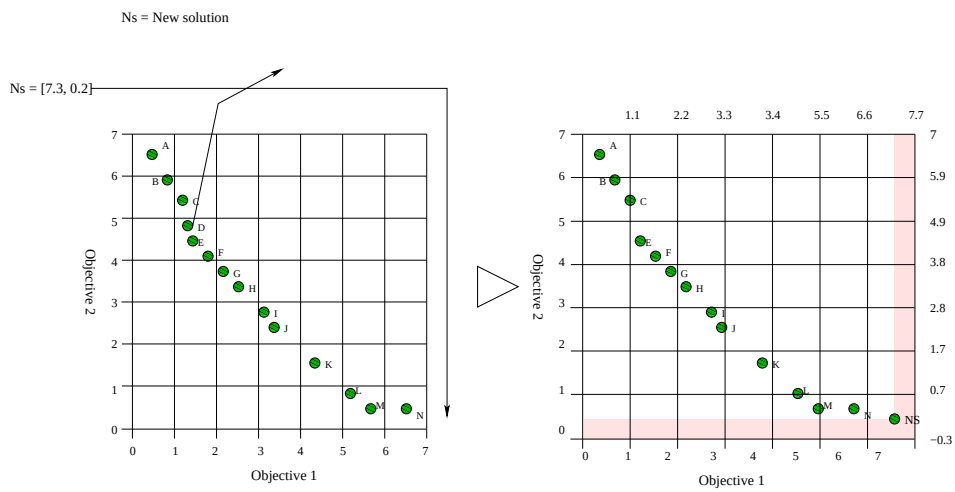

Figure 5.2: Graphical representation of the insertion of a new element in the adaptive grid when this lies outside the previous boundaries of the grid.

(see [71] for a detailed complexity analysis). The only exception would be if the grid had to be updated at each generation. In such case, the computational complexity of the adaptive grid would be the same as niching (i.e., $O(N^2)$ where N is the population size).

The adaptive grid is used to distribute in a uniform way the largest possible number of hypercubes. In order to achieve this goal, it is necessary to provide and obtain certain information which is problem-dependent (e.g., the number of grid subdivisions).

Next, a brief description of the information needed is provided, indicating in each case how this information is obtained:

Grid boundaries: These are the dimensions of the region that the grid must cover. Such dimensions are defined by the best and worst fitness values existing in the current contents of the grid. The upper bound of each component of this region is given by equation 5.1 (Min_i, is the lowest fitness for objective i), and the lowest is defined by equation 5.2 (Max_i is the largest fitness for objective i).

$$Min_i \leftarrow f_i(x) \mid \vec{x}^* \in \mathcal{P} \; and \; \neg\exists \; \vec{y}^* \in \mathcal{P} \; s.t. \; f_i(\vec{y}*) < f_i(\vec{x}^*) \tag{5.1}$$

$$Max \leftarrow f_i(x) \mid \vec{x}^* \in \mathcal{P} \; and \; \neg\exists \; \vec{y}^* \in \mathcal{P} \; s.t. \; f_i(\vec{y}*) > f_i(\vec{x}^*), \; i = 1, 2, \ldots, k \tag{5.2}$$

where:

k = Total number of objectives.

P^* = Pareto optimal set found.

A modification in the adaptive grid in order to reduce the number of updates required was proposed in [14]. The modifications consisted in adding some extra room to each grid component. Such extra room is equal to the dimensions of a hypercube, as shown in Figure 5.3 (the length of each hypercube component is divided by two to allow some extra room in both grid boundaries).

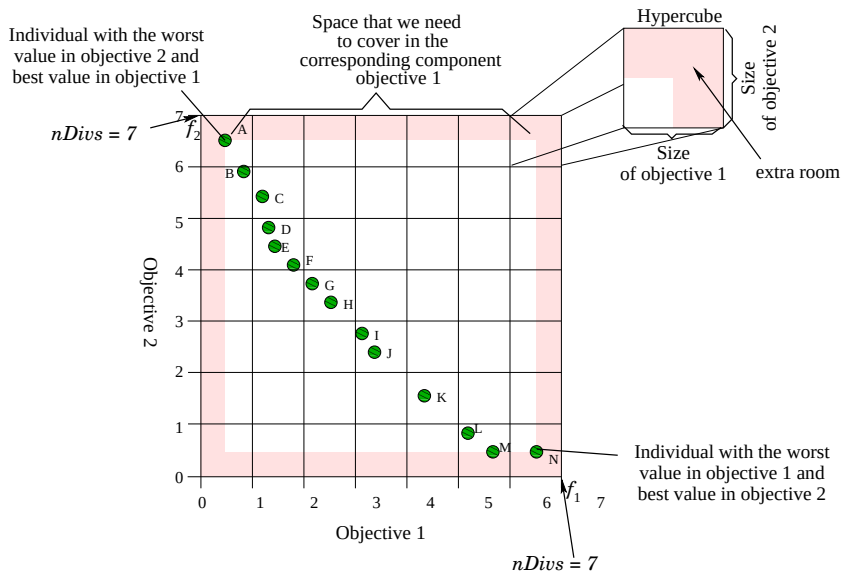

Figure 5.3: Graphical representation of the adaptive grid. In this case, we used two objective functions for ease of understanding (assuming minimization).

Hypercube dimensions: The subdivision (component) size i of each region $sizeDiv_i$) is obtained using the following expression:

$$sizeDiv_i \leftarrow \frac{Max_i - Min_i}{nDivs - 1};\ i = 1, 2, \ldots, k \tag{5.3}$$

where: k = Total number of objectives.
$nDivs$ = Number of subdivisions of the objective function space.

Region Identification The region ($LOCs$) to which the solution s belongs is obtained using:

$$LOCs = \sum_{i=1,j=k}^{k,1} \frac{f(x)_i - Min_i}{sizediv_i} * ndivs^j \tag{5.4}$$

where:
k = Number of objectives.
$f(\vec{x})_i$ = Fitness of the individual $\vec{x}$ with respect to objective i.
$sizediv_i$ = Subdivision size for objective i.
and $ndivs$= Number of subdivisions used.

The pseudo-code of the adaptive grid is shown in Algorithm 22.

Algorithm 22 Adaptive grid

1: **input** $\vec{x}$
2: $y \leftarrow z|z$
3: **for all** $i \in 1, ..., m$ **do**
4: $\quad b_i \leftarrow \frac{\log f_i}{\log(1+\epsilon)}$
5: **end for**
6: $b \leftarrow (b_1, ..., m)$
7: **OUTPUT** $b\{$box, index vector$\}$

5.3.2 ϵ-dominance

This is a relaxed form of dominance proposed by Laumanns et al. [80]. The so-called ϵ-Pareto set is an archiving strategy that maintains a subset of generated solutions. It guarantees convergence and diversity according to well-defined criteria i.e., ϵ-dominance and ϵ-Pareto optimality. The general idea is to divide the objective space into boxes of size ϵ. Each box can be interpreted as a geographical region that contains a single solution. The approach accepts a new solution into the ϵ-Pareto set if 1) it is the only solution in the box which it belongs to, 2) it dominates to other(s) solution(s) or 3) it competes against other non-dominated solution inside the box, but it is closer to the origin vertex of the box. This algorithm is very attractive both from a theoretical and from a practical point of view. However, in order to achieve the best performance, it is necessary to provide the size of the box (the ϵ parameter) which is problem-dependent. Algorithm 23 shows the update function for the ϵ-Pareto set, and Algorithm 24 shows the function box algorithm.

Algorithm 23 Update function for the ϵ-Pareto set

x

1: **input** A, f
2: $D \leftarrow \{f' \in A | box(f) \prec box(f')\}$
3: **if** $D \neq 0$ **then**
4: $\quad A' \leftarrow A \cup \{f\} \backslash D$
5: **else if** $\exists f' : (box(f') = box(f) and f \prec f')$ **then**
6: $\quad A' \leftarrow A \cup \{f\} \backslash \{f'\}$
7: **else if** $\exists f' : (box(f') = box(f) or f \prec f')$ **then**
8: $\quad A' \leftarrow A \cup \{f\}$
9: **else**
10: $\quad A' \leftarrow A$
11: **end if**
12: **output** A'

Algorithm 24 function box

1: **input** f
2: **for all** $i \in 1, ..., m$ **do**
3: $\quad b_i \leftarrow \frac{\log f_i}{\log(1+\epsilon)}$
4: **end for**
5: $b \leftarrow (b_1, ..., m)$
6: **OUTPUT** $b\{$box, index vector$\}$

5.3.3 Hyper-plane distribution

The core idea of this proposal is to perform a good distribution of the hyper-plane space defined by the minima (assuming minimization) from the objectives, and use such distribution to select a representative subset from the whole set of non-dominated solutions.

The complete pseudo-code is described in Algorithm 25. In words, the algorithm works as follows: First it accepts a set of non-dominated vectors and a number n of solutions of the desirable subset as its input. Then, the algorithm selects those vectors which have the minimum and maximum value of each objective, and it groups them into two sets, the minima set (called MIN), and the maxima set (called MAX). Using MIN, the algorithm creates a hyper-plane, and distributes its space into $n-1$ fixed-size sub hyper-planes. After that, it computes lines on each subdivision; such lines are perpendicular to the hyper-plane. Finally, the algorithm returns the closest vectors to each line.

Algorithm 25 Hyper-plane distribution

1: **input** $\mathcal{A}, n$
2: **for all** $i \in 1, ..., o$ **do**
3: $\quad \mathcal{MIN} \leftarrow \mathcal{MIN} \cup f(x) \mid \vec{x}^* \in \mathcal{A}$ *and* $\neg\exists\ \vec{y}^* \in \mathcal{A}$ *s.t.* $f_i(\vec{y}*) < f_i(\vec{x}^*)$
4: $\quad \mathcal{MAX} \leftarrow \mathcal{MAX}\ f(x) \mid \vec{x}^* \in \mathcal{A}$ *s.t.* $\neg\exists\ \vec{y}^* \in \mathcal{A} \mid f_i(\vec{y}*) > f_i(\vec{x}^*)$
5: **end for**
6: **for all** $i \in 1, ..., o$ **do**
7: $\quad \Delta_i \leftarrow \frac{\mathcal{MAX}_i - \mathcal{MIN}_i}{n-1}$
8: $\quad DIV_0^i \leftarrow \mathcal{MIN}_0$
9: **end for**
10: **for all** $i \in 2, ..., n$ **do**
11: $\quad DIV_i \leftarrow DIV_{i-1} + \Delta_i$
12: **end for**
13: $N \leftarrow |\frac{\mathcal{MIN}_2^2 - \mathcal{MIN}_1^2}{\mathcal{MIN}_2^1 - \mathcal{MIN}_1^1}|$
14: $m \leftarrow \frac{-1}{N}$
15: **for all** $x \in A$ **do**
16: $\quad$ **for all** $y \in DIV$ **do**
17: $\quad\quad b \leftarrow \text{distance}(x, y, m)$
18: $\quad\quad B \leftarrow\ B \cup b \mid \neg\exists\ b^* = distance(x*, y, m) \mid b^* < b$
19: $\quad$ **end for**
20: **end for**
21: **OUTPUT** B

Algorithm 26 distance from a point x to a straight

1: **input** x, y, m
2: $d \leftarrow (\frac{x_2 - y_1 - m \times (x_1 - y_2)}{\sqrt{(1.0 + m^2)}})$
3: **OUTPUT** d

In Figure 5.4 we can see an example that aims to clarify the algorithm's description. In this example, two objective functions are used. Five non-dominated solutions need to be selected. So, the hyper-plane (a line in this case) formed by the minima of objective 1 and objective 2 is divided into 4 line segments. Then, each vertex is projected towards the Pareto front. Finally, the solutions closest to those projected points are selected.

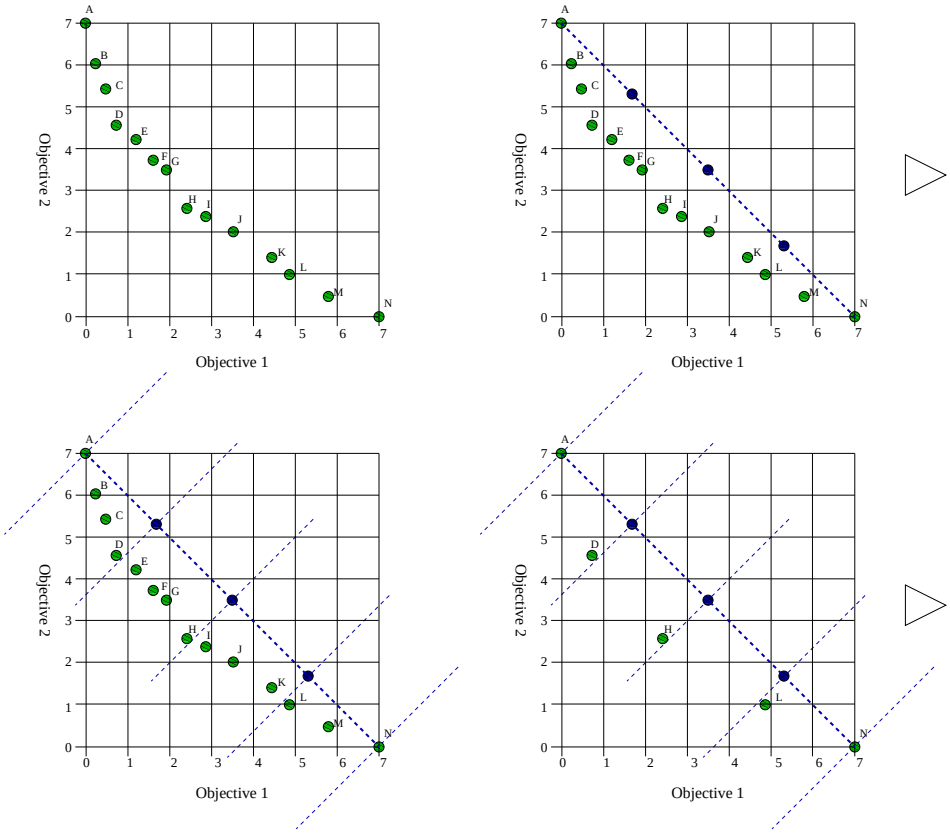

Figure 5.4: Graphical representation of the hyper-plane distribution.

5.3.4 Experiment 1

Algorithm 21 described before was independently executed 30 times in several test functions. The parameters of the algorithm were: C1=1.4962, C2= 1.4962, W=0.7298, 40 particles, 50 cycles of execution (150 for Kursawe's test function).

Such parameters were empirically derived after performing an exhaustive set of experiments.

These parameters result in a total of 2,000 fitness function evaluations (6,000 fitness function evaluations for Kursawe's test function). The fitness function evaluations were increased for this experiment since we wanted to measure an specific aspect (the spread of solutions) rather than assessing its performance.

The three implementations previously described to select the best distributed non-dominated individuals were executed on the non-dominated vectors previously found. The three algorithms were intended to select 50 non-dominated solutions. The scheme based on the ϵ-Pareto set needed an extra parameter (ϵ). This parameter was calculated in each test function as follows: Algorithm 21 was executed using a total number of iterations of 200; the rest of the parameters remained the same as before. Then, the ϵ values were manually fine-tuned to find an average of 40 non-dominated solutions in each of the 30 executions.

Since, there is no metric specifically designed to measure the distribution of a non-dominated set over the true Pareto front, we tested the Spacing [108] metric. Such a metric measures how well-distributed the non-dominated vectors are themselves.

In order to measure the coverage of each algorithm, the Hyper Volume metric [130] was used. This metric was applied on each algorithm's execution as follows: Since the vectors to which the distribution algorithms were applied are the same, we intended to use this metric to measure if an algorithm loses the solutions that are on each of the function's extremes.

Tables 5.1 and 5.2 show the results obtained by the three algorithms tested using the Hyper Volume and the Spacing metrics, respectively. We think that graphical results would say best the behavior of each approach. In Figures 5.5, 5.6 and 5.7 we can see a graphical comparison among the three algorithms for the test functions ZDT1, ZDT2, and ZDT3. The output's execution shown was randomly selected. From Figures 5.5, 5.6 and 5.7, it is easy to see that the three algorithms had problems in disconnected test functions. The adaptive grid did not produce a very good distribution of the solutions. ϵ-dominance presents several disadvantages in convex (see Figure 5.5) and disconnected (see Figure 5.7) Pareto fronts. Its main advantage is its efficiency. However its performance depends on the choice of the ϵ parameter which is difficult to set *a priori*.Hyper-plane distribution was the algorithm which could generate the best distribution of solutions. Also, it did not lose the extremes of the Pareto front.

Func.	Alg.	Adap-Grid	ϵ-P	H-D
ZDT1	**Adap-Grid**	———	0.0027987513	0.002989896
	ϵ-P	0.0076577354	———	0.003893796
	H-D	0.021349466	0.0174119706	———
ZDT2	**Adap-Grid**	———	0.0015273106	0.001391825
	ϵ-P	0.0039846446	———-	0.0020386273
	H-D	0.003689114	0.00187874	———
ZDT3	**Adap-Grid**	———	0.0011442646	0.0011813402
	ϵ-P	0.0017984955	———	0.0013139349
	H-D	0.0726156017	0.0720935426	———
ZDT6	**Adap-Grid**	———	0.0099886303	0.0169291256
	ϵ-P	0.0068796153	———	0.0188726704
	H-D	0.0062245308	0.0102926341	———
Kursawe	**Adap-Grid**	———	0.1248326433	0.1797871
	ϵ-P	0.4396478	———	0.2516026666
	H-D	0.3867131	0.1434585766	———
Deb	**Adap-Grid**	———	0.00329479	0.0032775870
	ϵ-P	0.0076676150	———	0.0037843980
	H-D	0.0075041325	0.0039599973	———
Deb2	**Adap-Grid**	———	0.04056185	0.05473879
	ϵ-P	0.047844055	———	0.065993434
	H-D	0.0490543256	0.0533195066	———

Table 5.1: Comparison of results of the approaches used to provide a good distribution of solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to the Hyper Volume metric.

Function	Adap-Grid	ϵ-P	H-D
ZDT1	**0.029328003**	0.036398970	0.032798373
ZDT2	**0.015502297**	0.018568336	0.092338194
ZDT3	0.042628383	0.059142866	**0.038913323**
ZDT4	**10.97824470**	14.87368026	23.79822963
ZDT6	0.373819456	**0.024635113**	0.045158286
Kursawe	**0.224578100**	0.239040166	0.225306533
Deb	**0.028016396**	0.031930420	0.032364046
Deb2	0.257862066	0.391271666	**0.185809133**

Table 5.2: Comparison of results of the approaches used to proved a good distribution of solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to Spacing metric.

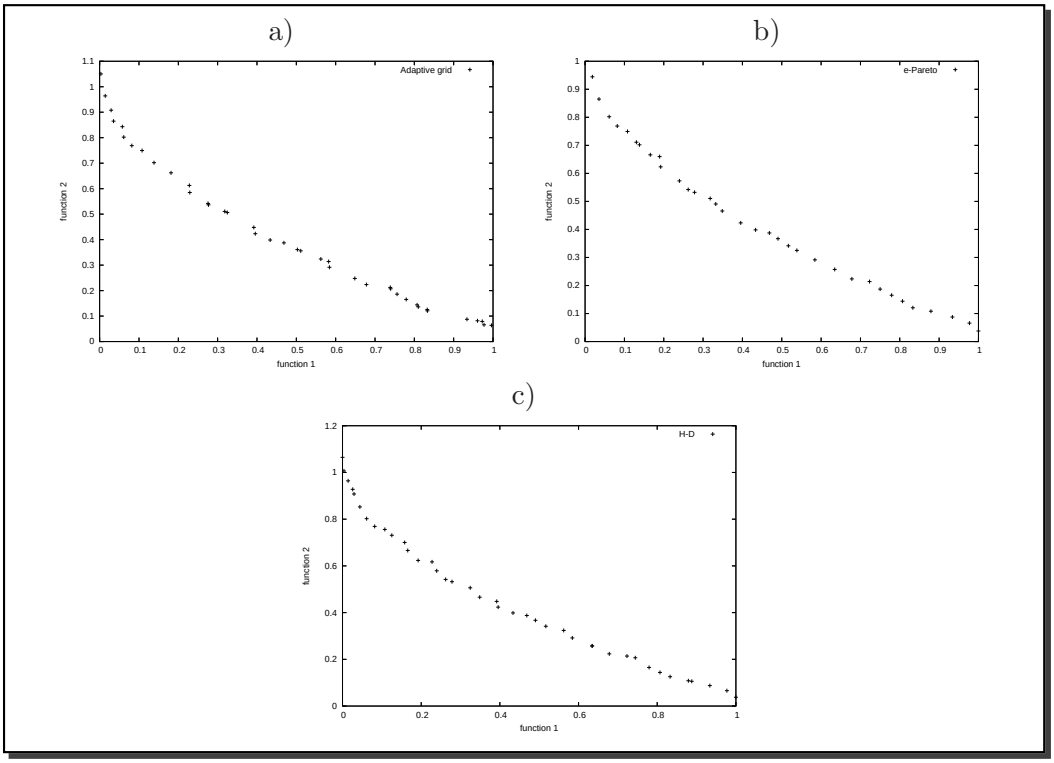

Figure 5.5: Pareto fronts produced by a) adaptive grid, b) ϵ-Pareto, and c) hyper-plane distribution, for the ZDT1's test function.

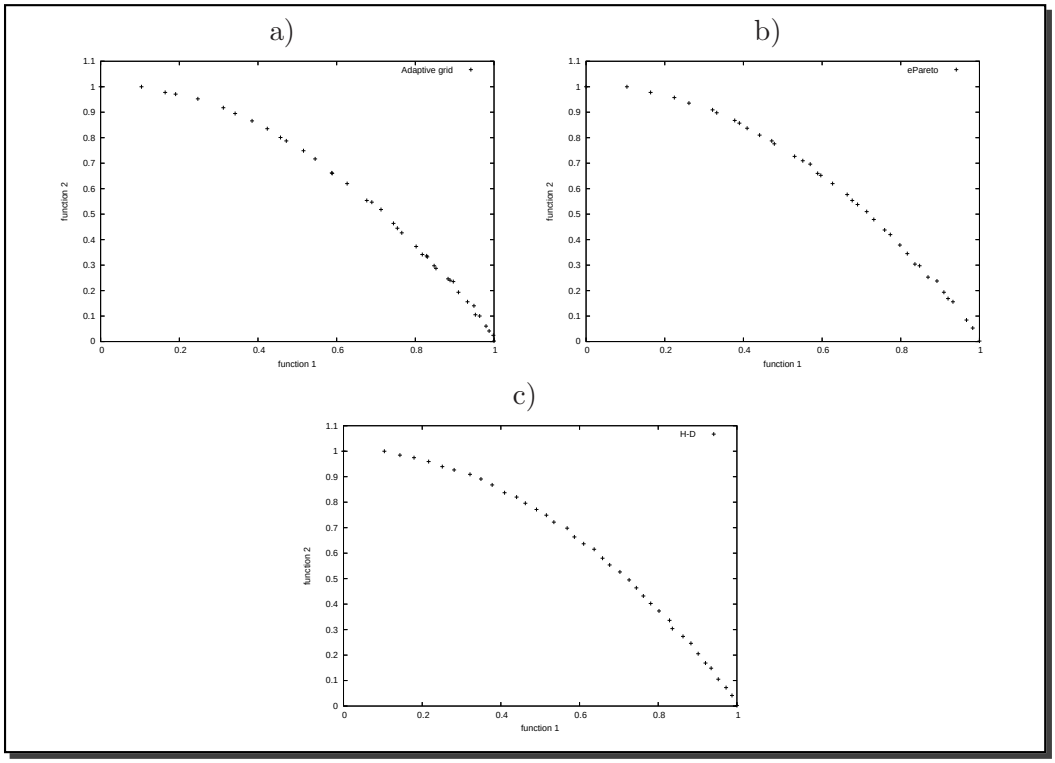

Figure 5.6: Pareto fronts produced by a) adaptive grid, b) ϵ-Pareto, and c) hyper-plane distribution, for the ZDT2's test function.

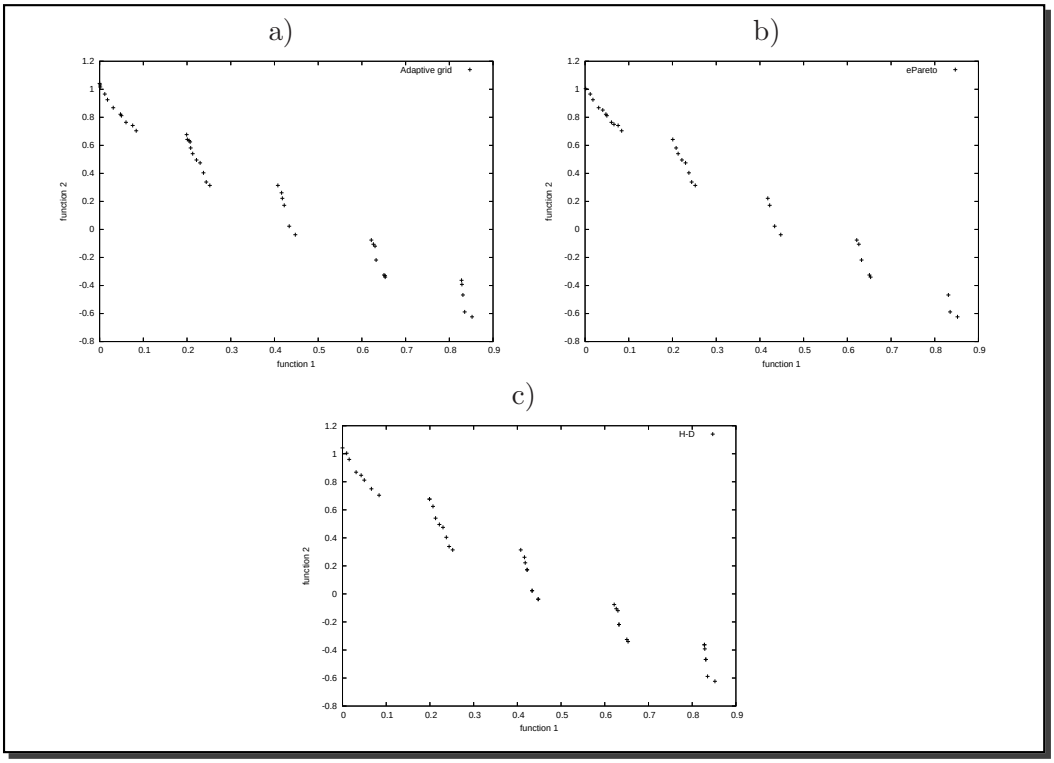

Figure 5.7: Pareto fronts produced by a) adaptive grid, b) ϵ-Pareto, and c) hyper-plane distribution, for the ZDT3's test function.

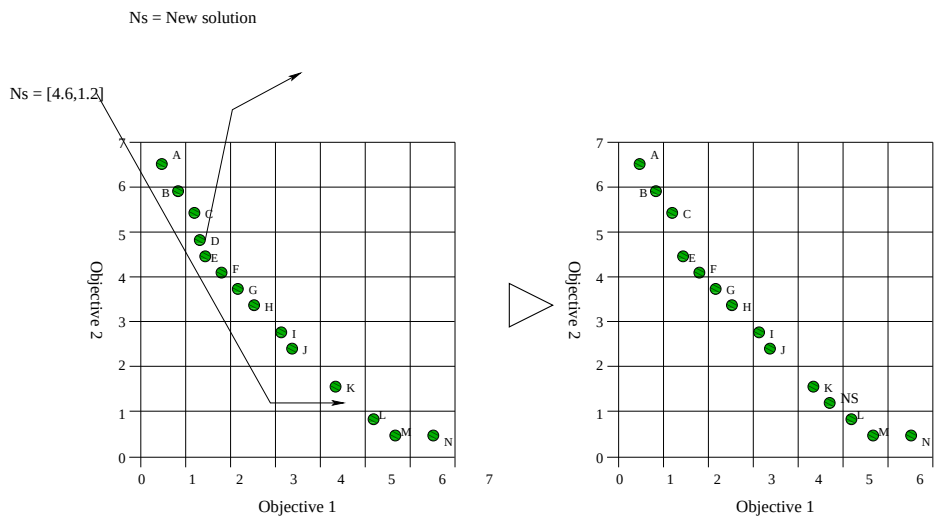

Figure 5.8: Graphical representation of the insertion of a new element in the adaptive grid when the individual lies within the current boundaries of the grid.

5.3.5 Experiment 2

Think of the following scenario: Algorithm 21 has discovered a set of n non-dominated solutions. Most of the n solutions are grouped very closely of each other, while the remaining solutions are distributed along the known Pareto front (see Figure 5.8). It seems obvious that at each generation, the algorithm is more likely to select more random solutions (to be used as *gbest*) from the most crowded solutions. In our last experiment (see Section 5.3.4), we introduced three algorithms to select a subset of well distributed non-dominated solutions. However, these approaches did not contribute to improve the performance of the algorithm in terms of efficacy. One could expect that if the number of non-dominated solutions is reduced at each generation to a well-distributed subset, there will be a faster improvement of solutions, since the followers will no longer spend more of their search efforts exploring a single region.

The aim of this experiment is to replace at each generation the set of non-dominated solutions (GBEST set) with a bounded and well distributed subset. The pseudocode of this proposal is given in Algorithm 27. To analyze the behavior of this algorithm, the test functions previously tested were used again. The algorithm using each of the schemes previously described to select distributed solutions was run for a total of 30 executions on each test function. Since, the aim of this experiment was to observe if the inclusion of an algorithm to preserve distribution improves the convergence of the MOPSO algorithm, the results were compared with respect to those solutions obtained by the algorithms used in experiment 1 (Section 5.3.4), using the Inverted Generational Distance described in Chapter 3. The aim of this metric is to measure the distance from the true Pareto front towards the Pareto front obtained by the approach. In Table 5.3 we can see that the Hyper-plane Distribution approach outperform most of the time to the Adaptive Grid and the ϵ-Dominance approaches. It seems that the Hyper-plane Distribution outperforms the ϵ-Dominance approach mainly because of its property of selecting a fixed number of solutions

(while we can only estimate the total number of solutions when using ϵ-Dominance). Table 5.4 shows the average of non-dominated solutions found in the last generation by each approach.

Algorithm 27 MOPSO - Step 2

Require: $nparticles, dimensions, W, C1$ and $C2$

$\vec{gbest} \leftarrow \vec{x_0}$
$GBEST \leftarrow \emptyset$
for $i = 0$ to $nparticles$ **do**
  $\vec{pbest_i} \leftarrow \vec{x_i} \leftarrow initialize_randomly()$
  $fitness_i \leftarrow f(\vec{x_i})$
  $velocity_{id} \leftarrow 0$
  if $\neg\exists\ \vec{y}^* \in GBEST \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness_i})$ **then**
    $GBEST \leftarrow GBEST \cup \vec{x_i}$
  end if
end for
repeat
  for $i = 0$ to $nparticles$ **do**
    if flip(0.5) **then**
      $gbest \leftarrow GBEST_{U(0,|GBEST|)}$
    else
      $gbest \leftarrow y | y \in GBEST \cup \nexists\ z \in GBEST | z - x_i < y - x_i$
    end if
    for $d = 0$ to $ndimensions$ **do**
      $velocity_{id} \leftarrow W \times velocity_{id} + C_1 \times U(0,1) \times (pbest_{id} - x_{id}) + C_2 \times U(0,1) \times (gbest - x_{id})$
      $x_{id} \leftarrow x_{id} + velocity_{id}$
    end for
    $fitness_i \leftarrow f(\vec{x_i})$
    if $fitness_i$ is $\sim$ to $f(\vec{pbest_i})$ **then**
      $\vec{pbest_i} \leftarrow \vec{x_i}$
    end if
    if $\neg\exists\ \vec{y}^* \in GBEST \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness_i})$ **then**
      $GBEST \leftarrow GBEST \cup \vec{x_i}$
    end if
  end for
  $GBEST \leftarrow approach - to - distribute - solutions(GBEST)$
until Termination criterion

	Approach					
Function	**Adap-Grid**	**Adap-Grid-b**	**ϵ-P**	**ϵ-P-b**	**H-D**	**H-D-b**
ZDT1	0.002713902	0.002855236	0.002544098	0.001492487	0.002549523	0.002029507
ZDT2	0.030849775	0.031790172	0.030796822	0.033058091	0.026445812	0.023006512
ZDT3	0.007221936	0.007237485	0.007233594	0.005816599	0.007181607	0.005786199
ZDT4	2.837542600	2.763645000	2.837542600	2.903810666	0.050004300	2.715304000
ZDT6	0.000209901	0.000224374	0.000231870	0.000186435	0.000389685	0.000197410
Kursawe	0.010552736	0.009648260	0.008594080	0.008716252	0.007743489	0.007975956
Deb	0.001043389	0.001454523	0.000958621	0.001606147	0.000857324	0.000475256
Deb2	0.009588622	0.009424353	0.009587181	0.009346694	0.009581715	0.009427080

Table 5.3: Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to the Inverted Generational Distance metric.

	Approach		
Function	**Adap-Grid-b**	**ϵ-P-b**	**H-D-b**
ZDT1	39	37	40
ZDT2	24	19	27
ZDT3	38	29	37
ZDT4	2	2	2
ZDT6	40	42	39
Kursawe	40	33	39
Deb	40	36	40
Deb2	39	34	37

Table 5.4: Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to the Inverted Generational Distance metric.

5.4 Maximizing the Spread

It is known that several difficult multiobjective optimization problems have a disconnected decision variable space. This issue is particularly important when using PSO, because it could be the case that a particle tries to follow a leader that resides in a disconnected region away from it. In this case, a lot of search effort would be wasted and the algorithm might not be able to converge to the true Pareto front of the problem. Other problem presented in disconnected fronts is the possibility to converge only to a single disconnected region. Algorithm 27 presents these problems. A natural way to solve these problems is the use of neighborhoods (i.e. groups of particles each of which overflies a different region) or several swarms. Kursawe's test function clearly illustrates this situation. Figure 3.1 shows the Pareto optimal set (up) and the Pareto front (down) of this test function.

Algorithm 21 was modified to perform a subselection of the well distributed non-dominated solutions. Four subset sizes were selected ($n = 40$, 20, 10 or 5). The methodology for selecting is the following: First we reduce the GBEST set to a well distributed non-dominated bounded subset. Next, we randomly select a solution from GBEST. Finally, we select its $n - 1$ closer solutions. These solutions will be the GBEST solutions used by the algorithm. The aim is to explore a different region of the non-dominated set at each generation.

The algorithm was also modified to use a different number of particles m ($m = 40$, 20, 10, 5). The particles are in an inner loop, where they iterate to a total of $40/m$ cycles. Table 5.5 summarizes the average value of the inverted generational distance applied to each of combinations of number of particles and size of the GBEST subset.

		Gbest set			
Function	**Part.**	**40**	**20**	**10**	**5**
ZDT1	40	0.008740401	0.008766247	0.007857118	0.011171814
	20	0.007731990	0.007585521	0.011522186	0.011875909
	10	0.010431674	0.009868152	0.014293828	0.016896183
	5	0.020298789	0.020027336	0.020777738	0.028044723
ZDT2	40	0.039022653	0.039221400	0.039529012	0.037630835
	20	0.059041604	0.059020294	0.058229974	0.058419296
	10	0.083991033	0.083838263	0.084064560	0.088129146
	5	0.115027300	0.115547496	0.115645843	0.113714746
ZDT3	40	0.020925614	0.020307261	0.021428732	0.020545984
	20	0.019615418	0.017401076	0.020407710	0.023908489
	10	0.019973149	0.019807940	0.026367473	0.032713004
	5	0.025688653	0.026204582	0.033566385	0.046927730
ZDT4	40	3.565314333	3.565314333	3.565314333	3.373635000
	20	3.301662000	3.301662000	3.301662000	3.216313000
	10	2.829766333	2.829766333	2.829766333	.2.792272666
	5	3.055342000	3.055342000	3.055342000	3.033530333
ZDT6	40	0.000423042	0.000396722	0.000388150	0.000317162
	20	0.003980978	0.003902585	0.003923901	0.003908334
	10	0.003867982	0.003834408	0.003845233	0.007505861
	5	0.029082076	0.029054262	0.025504887	0.018247513
Kursawe	40	0.009770807	0.010016194	0.012020978	0.011947436
	20	0.009839589	0.009682690	0.010303000	0.010419360
	10	0.012550338	0.011943884	0.015411651	0.014242191
	5	0.020899986	0.023561950	0.017872546	0.020147175
Deb	40	0.003670332	0.004207282	0.005605497	0.009291181
	20	0.001859000	0.003408800	0.004473308	0.007287329
	10	0.006968124	0.007708419	0.007584672	0.008808798
	5	0.016466266	0.015835591	0.013815201	0.018362347
Deb2	40	0.009501995	0.009556906	0.009569795	0.009311628
	20	0.009740814	0.010006768	0.009921843	0.009868808
	10	0.009809276	0.009951781	0.009904942	0.009715037
	5	0.009856391	0.009914928	0.009886514	0.009754674

Table 5.5: Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to Inverted Generational Distance metric.

5.4.1 Using Subswarms to Improve the Spread

In this experiment, several swarms (each with a fixed size) are used to solve the problem described above. Each swarm overflies an specific region of the non-dominated set (i.e., decision variable space), and has its own niche of particles and particle guides. To associate leaders within a swarm any clustering algorithm can be adopted. In the implementation of this algorithm, the hierarchical single-connected clustering algorithm [63] was used.

The appropriate selection of leaders is essential for the good performance of PSO when applied to multiobjective optimization problems. If the particle chooses an inappropriate leader (i.e., a leader who is too far away in the search space) then most of the flight will be fruitless because the particle will not be traversing promissory regions of the search space. In this algorithm, we propose to use not one but several swarms to avoid this type of problem.

The algorithm with sub-swarms is shown in Algorithm 28.

The complete execution process of our algorithm can be divided in two stages: initialization and flight.

At the first stage, every sub-swarm is initialized. Each sub-swarm creates and initializes its own particles and generates the leaders set among the particle swarm set by using Pareto ranking. In the second stage is where the algorithm performs its strongest effort.

First, it performs the execution of the flight of every swarm; next, it applies a clustering algorithm to put together the leading particles. This is performed until reaching a total of $GMax$ iterations. The execution of the flight of each swarm can be seen as an entire PSO process (with the difference that it will only optimize an specific region of the search space). First, each particle will select a leader to which it will follow. At the same time, each particle will try to outperform its leader and to update its position. If the updated particle is not ϵ-dominated by any member of the leaders set, then it will become a new leader. The execution of the swarm will start again until a total of $sgmax$ iterations is reached.

Once all the swarms have finished their flights, a clustering algorithm takes the control by grouping the closest particle guides into n_{swarms} swarms. These particle guides will try to outperform each swarm in the next iteration. This is mainly done by grouping the leaders of all the swarms into a single set, and then splitting this set among n_{swarms} groups (clustering is done with respect to closeness in decision variable space). Each resulting group will be assigned to a different swarm.

Since the right number of clusters for each problem is not known in advance, Table 5.6 summarizes the average values of inverted generational distance of 30 executions of the algorithm 28 on several test functions. We can see that there is not an unique number of clusters which performed best for all the test functions adopted.

Frans van den Bergh [116] discovered a potentially dangerous property in PSO: $if\ x_i = pbest_i = gbest_i$ then the value will depend only on $W_iV_i(t)$ (i.e. if the position of the particles coincides with gbest, then it will move away from the gbest if w_iv_i is non-zero). This may lead the algorithm to premature convergence (i.e. all the particles will converge to the gbest particle, which usually is a local minimum).

To determine how often this behavior occurred in our algorithm, we count how many times particle's position was the same that the gbest position in all the executions for all

Algorithm 28 MOPSO - step 3

Require: $nsubswarms, nparticles, dimensions, W, C1$ and $C2$

1: $\vec{gbest} \leftarrow \vec{x_0}$
2: $GBEST_{final} \leftarrow \emptyset$
3: **for** $s = 0$ to $nsubswarms$ **do**
4: $\quad GBEST^s \leftarrow \emptyset$
5: $\quad$ **for** $i = 0$ to $nparticles/nsubswarms$ **do**
6: $\quad\quad \vec{pbest}_i^s \leftarrow \vec{x_i^s} \leftarrow initialize_randomly()$
7: $\quad\quad fitness_i^s \leftarrow f(\vec{x_i^s})$
8: $\quad\quad velocity_{id}^s \leftarrow 0$
9: $\quad\quad$ **if** $\neg\exists\ \vec{y}^* \in GBEST^s \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness}_i^s)$ **then**
10: $\quad\quad\quad GBEST^s \leftarrow GBEST^s \cup \vec{x_i^s}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end for**
14: **repeat**
15: $\quad GBEST_{temp} \leftarrow \emptyset$
16: $\quad$ **for** $s = 0$ to $nsubswarms$ **do**
17: $\quad\quad$ **for** $i = 0$ to $nparticles$ **do**
18: $\quad\quad\quad$ **if** flip(0.5) **then**
19: $\quad\quad\quad\quad gbest \leftarrow GBEST^s_{U(0,|GBEST^s|)}$
20: $\quad\quad\quad$ **else**
21: $\quad\quad\quad\quad gbest \leftarrow y | y \in GBEST^s \cup \nexists\ z \in GBEST^s | z - x_i^s < y - x_i^s$
22: $\quad\quad\quad$ **end if**
23: $\quad\quad\quad$ **if** x^s is equal $gbest$ **then**
24: $\quad\quad\quad\quad$ turbulence (x^s)
25: $\quad\quad\quad$ **end if**
26: $\quad\quad\quad$ **for** $d = 0$ to $ndimensions$ **do**
27: $\quad\quad\quad\quad velocity_{id}^s \leftarrow W \times velocity_{id}^s + C_1 \times U(0,1) \times (pbest_{id}^s - x_{id}^s) + C_2 \times U(0,1) \times (gbest - x_{id}^s)$
28: $\quad\quad\quad\quad x_{id}^s \leftarrow x_{id}^s + velocity_{id}^s$
29: $\quad\quad\quad$ **end for**
30: $\quad\quad\quad fitness_i^s \leftarrow f(\vec{x_i^s})$
31: $\quad\quad\quad$ **if** $fitness_i^s$ is $\sim$ to $f(\vec{pbest}_i^s)$ **then**
32: $\quad\quad\quad\quad \vec{pbest}_i^s \leftarrow \vec{x_i^s}$
33: $\quad\quad\quad$ **end if**
34: $\quad\quad\quad$ **if** $\neg\exists\ \vec{y}^* \in GBEST^s \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness}_i^s)$ **then**
35: $\quad\quad\quad\quad GBEST^s \leftarrow GBEST^s \cup \vec{x_i^s}$
36: $\quad\quad\quad$ **end if**
37: $\quad\quad$ **end for**
38: $\quad\quad GBEST_{final} \leftarrow GBEST_{final} \cup GBEST^s$
39: $\quad$ **end for**
40: $\quad GBEST_{final} \leftarrow approach - to - distribute - solutions(GBEST_{final})$
41: $\quad$ group $GBEST_{final}$ into $nsubswarms$
42: $\quad$ **for** $s = 0$ to $nsubswarms$ **do**
43: $\quad\quad GBEST_s \leftarrow$ randomly select a group formed in last step
44: $\quad$ **end for**
45: **until** Termination criterion

	Clusters				
Function	**1**	**2**	**4**	**8**	**20**
ZDT1	0.009455527	0.009035658	0.006762290	0.004711227	0.003304528
ZDT2	0.034038000	0.040171617	0.031527797	0.034350482	0.034291232
ZDT3	0.024068386	0.027907103	0.015640700	0.013451196	0.015259319
ZDT4	2.896320000	2.928153000	3.058703000	3.441940666	3.453936666
ZDT6	0.000505861	0.000658532	0.000605209	0.000349758	0.000370221
Kursawe	0.036275200	0.034403426	0.035801300	0.032868496	0.030700593
Deb	0.086719610	0.008664331	0.006428156	0.005138733	0.005102338
Deb2	0.009684859	0.008918948	0.009142833	0.009558412	0.008997162

Table 5.6: Comparison of results of the approaches to maintain a good distribution of non-dominated solutions (adaptive grid, ϵ-dominance and hyper-plane distribution) with respect to Inverted Generational Distance metric.

the test functions. In Table 5.4.1 we can see the results of this experiment. We can see that this property is usually presented by the algorithm. Frans van den Bergh proposed a new parameter to address this issue. However, this proposal becomes hard to adopt in a multiobjective approach using Pareto ranking, since, it implies the use of several "best" solutions. Therefore, his online-adaptive approach will become very unstable. Instead of using this approach, we propose the use of the turbulence operator proposed in [115]. This operator has a behavior comparable to that of Frans van den Berg's proposal and it has been tested in a multiobjective scenario.

The turbulence consists of an alteration to the flight velocity of a particle.[2] This modification is performed in all the dimensions (i.e., in all the decision variables), such that the particle can move to a completely isolated region (something much more difficult to achieve by the mere use of the velocity adjustment formula described before). This mechanism aims to perturb the swarm as to avoid that the particles get trapped in local optima. The turbulence operator acts based on a probability that considers the current generation and the total number of iterations to be performed. The idea is to have a much higher probability to perturb the flight of the particles at the beginning of the search. Over time, this probability will be decreased as we progress in the search.

The turbulence can be seen as a mutation operator and it is based on the following expression:

$$temp = current_generation/total_generations \tag{5.5}$$

$$prob_{turbulence} = temp^{1.7} - 2.0 * (temp) + 1.0 \tag{5.6}$$

where $temp$ is used as a temporary variable, $current_generation$ is the current generation number, $total_generations$ is the total number of generations and $prob_{turbulence}$ refers to the probability of affecting the flight of a particle using the turbulence operator. The values used for this expression were empirically derived after a set of experiments.

After incorporating our methodology into the algorithm, we executed the experiment again. Table 5.8 summarizes the results obtained. After the modification we found that by

[2]This mechanism is inspired on [32].

Statistics	Particle GBest
Mean	104.8119227539
Best	2
Worst	729
St.dev.	110.5998308335
Median	74

Table 5.7: Statical results obtained from the counting where the particle was equal to its gbest.

	Clusters				
Function	**1**	**2**	**4**	**8**	**20**
ZDT1	0.004820567	0.005645242	0.004388993	0.0028531300	0.003191553
ZDT2	0.0347598000	0.013606517	0.017762557	0.0125824133	0.013187440
ZDT3	0.016337620	0.017927403	0.017630486	0.0087873916	0.011176582
ZDT4	4.536191000	3.445736000	3.276966000	3.7399510000	3.762421666
ZDT6	0.00333742300	0.000509384	0.000472352	0.0003349334	0.000431701
Kursawe	0.0695214000	0.041491300	0.043504603	0.0498330600	0.042813556
Deb	0.034178800	0.009745156	0.007927125	0.0071140263	0.008663098
Deb2	0.009801385	0.009445994	0.008891579	0.008950303	0.009510023

Table 5.8: Comparison of results of MOPSO with respect to Inverted Generational Distance metric.

using 8 subswarms, the algorithm exhibited its best performance in 5 out of 8 test functions. We think, that the use of this value can be beneficial most of the time. Therefore, we adopted it as the default value for the number of subswarms. It is important to note that this experiment was performed using 40 particles, which means that each subswarm will have 5 particles. This also has relevance, since, the authors of the PSO algorithm have claimed that 5 particles are enough to reach convergence in any problem

Algorithm	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.993475	0.0168072
NSGA-II	0.260602	———-	0.0796734
ϵ-MOEA	0.305217	1.11986	———-

Table 5.9: Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for the Kursawe's test function.

Algorithm	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.476471	0.323529
NSGA-II	0.932642	———-	0.756477
ϵ-MOEA	0.95	0.846875	———-

Table 5.10: Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Kursawe's test function.

5.5 MOPSO's Comparison of Results

In the following examples, the NSGA-II was run using a population size of 40, a crossover rate of 0.8 (uniform crossover was adopted), tournament selection, and a mutation rate of $1/N$, where N = number of variables (real representation was adopted), a distribution index of 15 for real-coded crossover, a distribution index of 20 for real-coded mutation. The ϵ-MOEA was run using a population size of 40, a crossover rate of 0.8 (uniform crossover was adopted), distribution index of 15 for real-coded crossover and distribution index of 20 for real-coded mutation. MOPSO used 40 particles, and a total of 8 swarms.

The total number of fitness function evaluations was set to $2,000$ for all the algorithms compared (50 generations).

5.5.1 Kursawe's Test Function

Figure 5.9 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO for Kursawe's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the median result with respect to the inverted generational distance metric. Tables 5.9, 5.10 and 5.11, show the comparison of results among the three algorithms considering hyper volume, two set coverage, inverted generational distance, and success counting metrics. It can be seen that the ϵ-MOEA performed best with respect to HV, TSC, IGD and SC by far. Our MOPSO was the algorithm which performed worst.

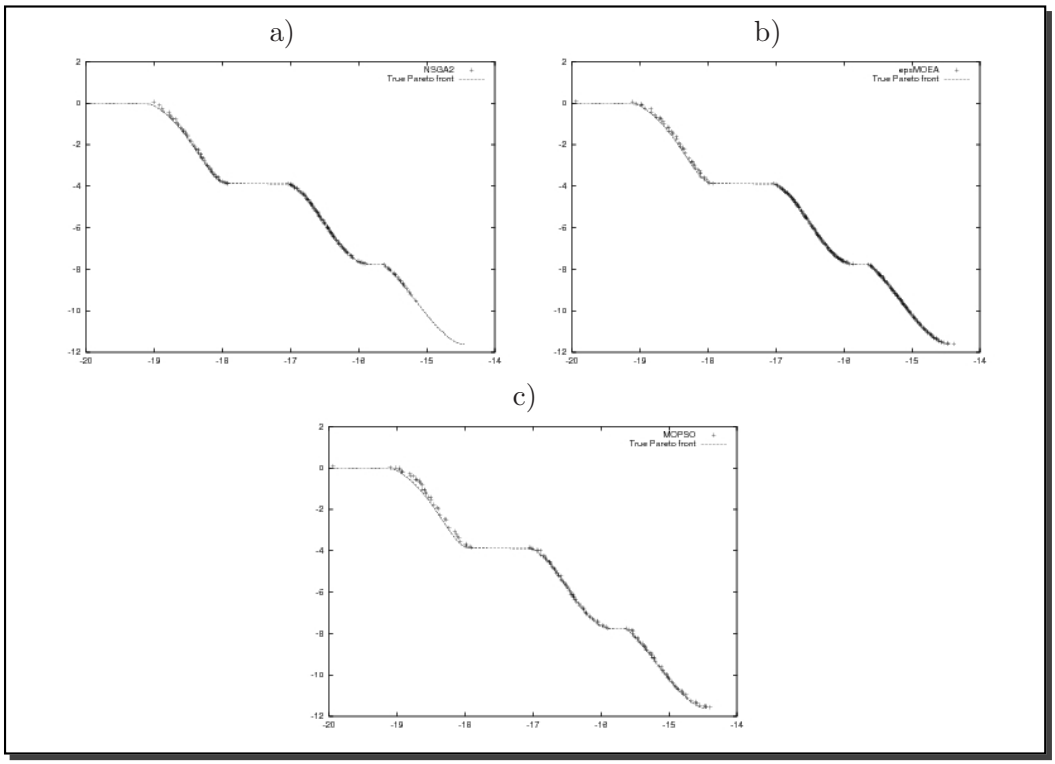

Figure 5.9: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Kursawe's test function.

Metric	Statistics	MOPSO	NSGA-II	ϵ-MOEA
IGD	**Best**	0.0102315	0.0588314	0.00405783
	Worst	0.0216605	0.117137	0.00973392
	Mean	0.0150202	0.0825022	0.00602655
	St. dev.	0.00327527	0.021286	0.00161515
	Median	0.0145841	0.078836	0.00546673
SC	**Best**	5	22	49
	Worst	0	8	33
	Mean	2.13333	15.5	40.5667
	St. dev.	1.35782	4.84056	5.69139
	Median	2	20	43

Table 5.11: Comparison of results of the MOPSO, NSGA-II and our ϵ-MOEA with respect to the Inverted Generational Distance (IGD) and Success Counting (SC) for Kursawe's test function.

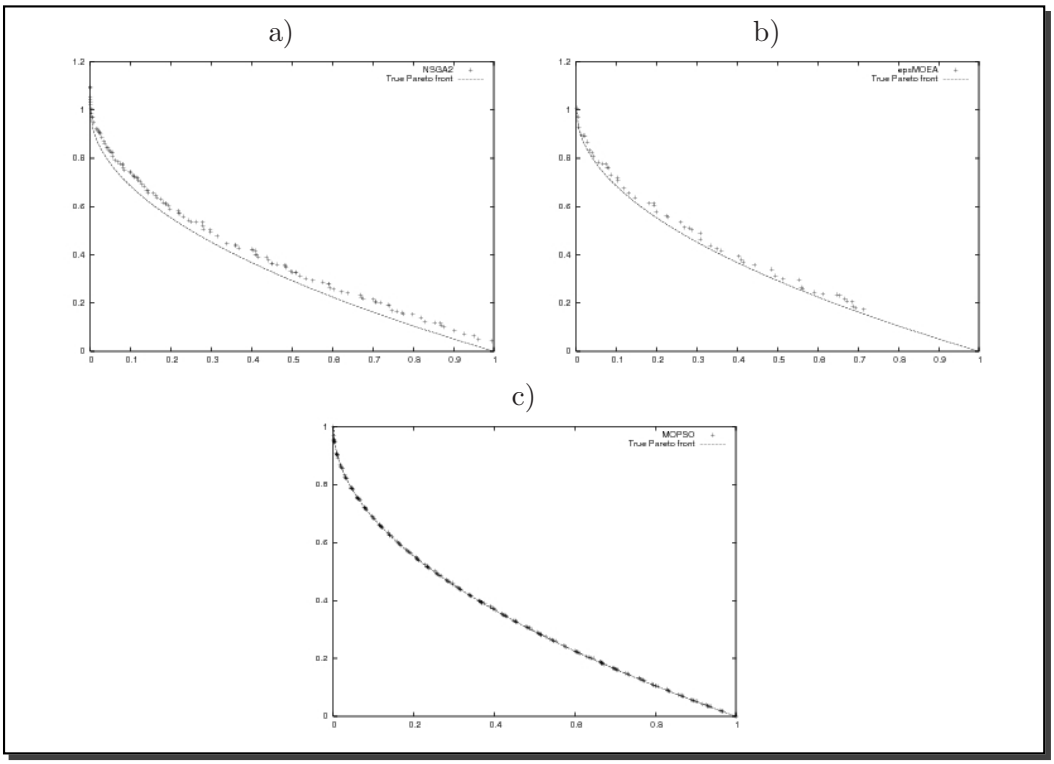

Figure 5.10: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for the ZDT1's test function.

Algorithm	**MOPSO**	**NSGA-II**	**ϵ-MOEA**
MOPSO	———-	0.0402313	0.0461881
NSGA-II	0	———-	0.0164115
ϵ-MOEA	5.1006e-05	0.0110925	———-

Table 5.12: Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT1's test function.

5.5.2 ZDT1' Test Function

Figure 5.10 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO for the ZDT1's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the median result with respect to the generational distance metric. Tables 5.12, 5.13 and 5.14, show the comparison of results among the three algorithms considering hyper volume, two set coverage, inverted generational distance and success counting metrics. It can be seen that our MOPSO performed best with respect to all the metrics proved.

Algorithm	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	1	1
NSGA-II	0	———-	0.359375
ϵ-MOEA	0.0535714	0.946429	———-

Table 5.13: Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT1's test function.

Metric	Statistics	MOPSO	NSGA-II	ϵ-MOEA
IGD	**Best**	0.000792374	0.00261635	0.00640436
	Worst	0.00101281	0.0169496	0.0379035
	Mean	0.000882652	0.00676222	0.0227843
	St. dev.	5.29644E-05	0.0045472	0.00851036
	Median	0.000874956	0.0049536	0.0240974
SC	**Best**	23	2	1
	Worst	5	0	0
	Mean	14.7667	0.0666667	0.0666667
	St. dev.	4.60647	0.365148	0.253708
	Median	17	0	0

Table 5.14: Comparison of results of the MOPSO, NSGA-II and our ϵ-MOEA with respect to Inverted Generational Distance (IGD) and Success Counting (SC) for ZDT1's test function.

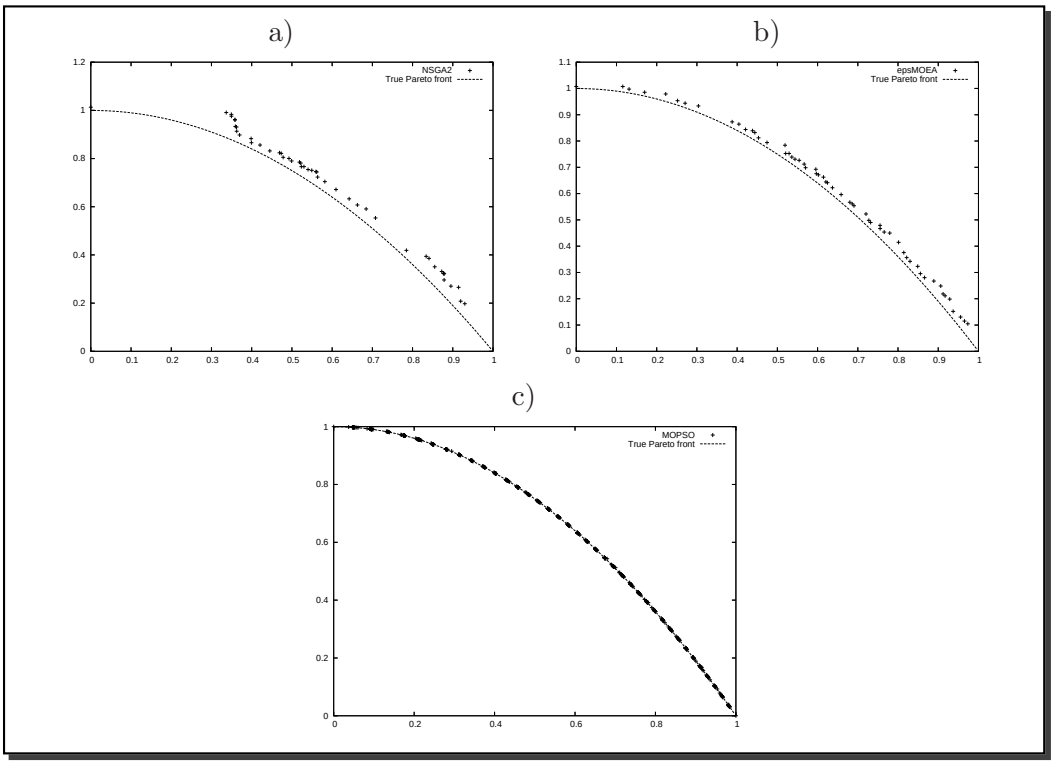

Figure 5.11: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for ZDT2's test function.

Algorithm	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———	0.0619725	0.0332476
NSGA-II	0	———	0.00145575
ϵ-MOEA	0	0.0277186	———

Table 5.15: Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT2's test function.

5.5.3 ZDT2's Test Function

Figure 5.11 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO for ZDT2's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the median result with respect to the generational distance metric. Tables 5.15, 5.16 and 5.17, show the comparison of results among the three algorithms considering hyper volume, two set coverage, inverted generational distance and success counting metrics. It can be seen that our MOPSO performed best with respect to HV, TSC, IGD, and SC by far.

Algorithm	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———	1	1
NSGA-II	0	———	0.276596
ϵ-MOEA	0	1	———

Table 5.16: Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT2's test function.

Metric	Statistics	MOPSO	NSGA-II	ϵ-MOEA
IGD	**Best**	0.000783849	0.00560298	0.00200895
	Worst	0.000873722	0.0573697	0.0514464
	Mean	0.000807004	0.0322927	0.0166433
	St. dev.	2.19975E-05	0.0188163	0.0143003
	Median	0.000800601	0.0294469	0.0130747
SC	**Best**	13	0	0
	Worst	41	0	0
	Mean	32.5333	0	0
	St. dev.	9.533	0	0
	Median	37	0	0

Table 5.17: Comparison of results of the MOPSO, NSGA-II and our ϵ-MOEA with respect to the Inverted Generational Distance (IGD) and Success counting (SC) for ZDT2's test function.

5.6 Conclusions

We have presented a new proposal to extend particle swarm optimization to handle multiobjective problems using sub-swarms, Pareto ranking and clustering techniques. The proposed approach was validated using the standard methodology currently adopted in the evolutionary multiobjective optimization community. The results indicate that our approach is a viable alternative since it outperformed some of the best multiobjective evolutionary algorithms known to date.

6

A Constraint-Handling Mechanism for PSO

6.1 Introduction

SINCE most real-world applications have constraints, multiobjective optimizers must have a mechanism to incorporate them. The Particle Swarm Optimization, like most other evolutionary algorithms, lacks an explicit mechanism to incorporate constraints.

This chapter presents a mechanism to handle constraints with a multiobjective particle swarm optimization algorithm However, the definition of a good constraint-handling mechanism requires that we step back and retake a single-objective version of PSO. This is necessary in order to validate our constraint-handling mechanism, since a well-established benchmark exists for that sake (unfortunately, we can't say the same regarding constrained multiobjective optimization). Thus we will proceed to develop a constraint-handling mechanism for single-objective PSO, following 3 steps. First, a criterion based on closeness of a particle to the feasible region in order to select a leader when dealing with constrained search spaces is implemented in a single-objective PSO (Section6.3). As a second step, and in order to know how competitive is the proposed approach, we tested our approach using a single-objective optimization benchmark proposed in [104] (Section 6.4). Our comparison of results indicates that the proposed approach is highly competitive with respect to two constraint-handling techniques representative of the state-of-the-art in the area. Next, our third step was to include the proposed approach is included into the MOPSO algorithm proposed in Chapter 4. The resulting approach is compared against other approaches representative of the state-of-the-art in the area. Results indicate that the approach is competitive solving constrained multiobjective optimization problems.

6.2 Related Work

When incorporating constraints into the fitness function of an evolutionary algorithm, it is particularly important to maintain diversity in the population and to be able to keep solutions both inside and outside the feasible region [19, 85]. Several studies have shown that, despite their popularity, traditional (external) penalty functions, even when used with dynamic penalty factors, tend to have difficulties to deal with highly constrained search spaces and with problems in which the constraints are active in the optimum [19, 76, 104]. The random generation of solutions until reaching the feasible region is another possible way to handle constraints (in fact, this approach has been used by some researchers when using the PSO algorithm [56, 57]). However, this sort of approach may become too expensive (computationally speaking) when dealing with highly constrained search spaces. Motivated by this fact, a number of constraint-handling techniques have been proposed for evolutionary algorithms [86, 19]. However, this topic has been only scarcely explored by PSO researchers [56, 12, 94, 57, 96].

6.3 Constrained Particle Swarm Optimization

Algorithm 29 CPSO Algorithm

1: $\vec{gbest} \leftarrow \emptyset$
2: **for** $i = 0$ to $nparticles$ **do**
3: $\vec{pbest}_i \leftarrow \vec{x_i} \leftarrow initialize_randomly()$
4: $fitness_i \leftarrow f(\vec{x_i})$
5: **if** $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{gbest}))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{gbest}) \cup fitness_i < f(\vec{gbest}))$ **then**
6: $\vec{gbest} \leftarrow \vec{x_i}$
7: **end if**
8: **end for**
9: **repeat**
10: **for** $i = 0$ to $nparticles$ **do**
11: **for** $d = 0$ to $ndimensions$ **do**
12: $velocity_{id} \leftarrow W \times velocity_{id} + C_1 \times U(0,1) \times (pbest_{id} - x_{id}) + C_2 \times U(0,1) \times (gbest - x_{id})$
13: $x_{id} \leftarrow x_{id} + velocity_{id}$
14: **end for**
15: turbulence (x_{id})
16: $fitness_i \leftarrow f(\vec{x_i})$
17: **if** $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{pbest}_i))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{pbest}_i) \cup fitness_i < f(\vec{pbest}_i))$ **then**
18: $\vec{pbest}_i \leftarrow \vec{x_i}$
19: **end if**
20: **if** $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{gbest}))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{gbest}) \cup fitness_i < f(\vec{gbest}))$ **then**
21: $\vec{gbest} \leftarrow \vec{x_i}$
22: **end if**
23: **end for**
24: **until** Termination criterion

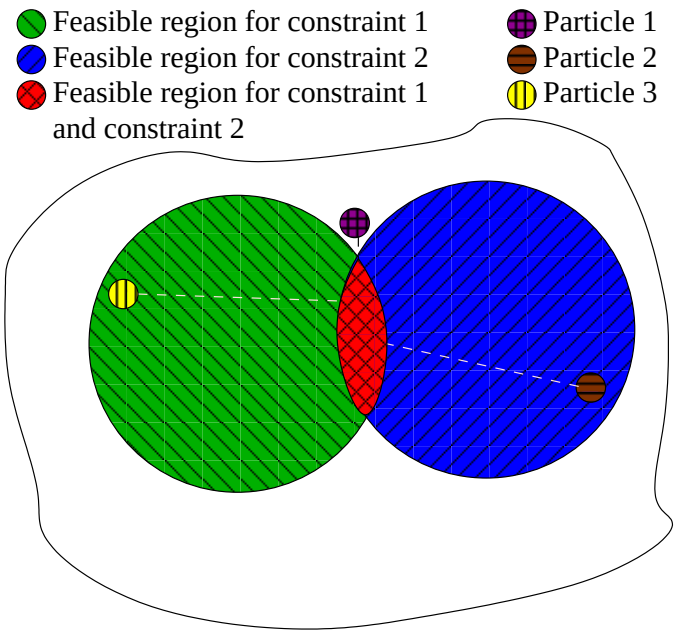

Figure 6.1: Graphical representation of our example that explains the constraint-handling mechanism incorporated into our PSO algorithm.

Figure 29 shows the PSO algorithm adopted for our study.

The algorithm is basically a simple PSO implementation, except for three aspects: the way in which the velocity is computed, the turbulence operator (see Section 5.4.1) and the mechanism adopted to handle constraints. These aspects are discussed in the following subsections.

6.3.1 Mechanism to Handle Constraints

The mechanism that we propose in this chapter to handle constraints is applied when selecting a leader. What we did was to perform a small change in the fitness function such that if we compare two feasible particles, the particle that has the highest fitness value wins. If one of the particles is infeasible and the other one is feasible, then the feasible particle wins. If both particles compared are infeasible, then the particle that has the lowest value in its total violation of constraints (normalized with respect to the largest violation of each constraint achieved by any particle in the current population) wins. The idea is to choose as a leader to the particle that, even when infeasible, lies closer to the feasible region. To understand better this idea, let's consider the following example:

Let's consider 3 particles and 2 constraints: **particle 1** violates in 30 units the first constraint and in 40 units the second constraint. **Particle 2** does not violate the first constraint, but it violates in 100 units the second constraint. Finally, **particle 3** violates in 130 units the first constraint, but it does not violate the second constraint. Furthermore, with respect to the total population, the largest violation of the first constraint is 200 and

the largest violation of the second constraint is 120. Thus, the fitness of **particle 1** is $30/200 + 40/120 = 0.48333$. The fitness of **particle 2** is $0 + 100/120 = 0.83333$. The fitness of **particle 3** is $130/200 + 0 = 0.65000$. So, **particle 1** has a better fitness than **particle 2** and **particle 3** (let's keep in mind that in this case, a smaller value indicates that the particle is closer to the feasible region), despite the fact that this particle violated the 2 constraints of the problem and the two other particles only violate one of them. This behavior is graphically depicted in Figure 6.1.

6.4 Test Functions

To evaluate the performance of the proposed approach we used the 13 test functions described in [104]. The test functions chosen contain characteristics that are representative of what can be considered "difficult" global optimization problems for an evolutionary algorithm. Their expressions are provided next.

1. **g01**:
   Minimize: $f(\vec{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$ subject to:

$$\begin{aligned} g_1(\vec{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\ g_2(\vec{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\ g_3(\vec{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\ g_4(\vec{x}) &= -8x_1 + x_{10} \leq 0 \\ g_5(\vec{x}) &= -8x_2 + x_{11} \leq 0 \\ g_6(\vec{x}) &= -8x_3 + x_{12} \leq 0 \\ g_7(\vec{x}) &= -2x_4 - x_5 + x_{10} \leq 0 \\ g_8(\vec{x}) &= -2x_6 - x_7 + x_{11} \leq 0 \\ g_9(\vec{x}) &= -2x_8 - x_9 + x_{12} \leq 0 \end{aligned}$$

   where the bounds are $0 \leq x_i \leq 1$ $(i = 1, \ldots, 9)$, $0 \leq x_i \leq 100$ $(i = 10, 11, 12)$ and $0 \leq x_{13} \leq 1$. The global optimum is at $x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ where $f(x^*) = -15$. Constraints g_1, g_2, g_3, g_4, g_5 and g_6 are active.

2. **g02**:
   Maximize: $f(\vec{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$ subject to:

$$\begin{aligned} g_1(\vec{x}) &= 0.75 - \prod_{i=1}^{n} x_i \leq 0 \\ g_2(\vec{x}) &= \sum_{i=1}^{n} x_i - 7.5n \leq 0 \end{aligned}$$

   where $n = 20$ and $0 \leq x_i \leq 10$ $(i = 1, \ldots, n)$. The global maximum is unknown; the best reported solution is [104] $f(x^*) = 0.803619$. Constraint g_1 is close to being active

($g_1 = -10^{-8}$).

3. **g03**:
   Maximize: $f(\vec{x}) = (\sqrt{n})^n \prod_{i=1}^{n} x_i$

   subject to:

   $h(\vec{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$

   where $n = 10$ and $0 \leq x_i \leq 1$ $(i = 1, \ldots, n)$. The global maximum is at $x_i^* = 1/\sqrt{n}$ $(i = 1, \ldots, n)$ where $f(x^*) = 1$.

4. **g04**:
   Minimize: $f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$
   subject to:
   $g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0$
   $g_2(\vec{x}) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0$
   $g_3(\vec{x}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0$
   $g_4(\vec{x}) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0$
   $g_5(\vec{x}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0$
   $g_6(\vec{x}) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0$

   where: $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$, $27 \leq x_i \leq 45$ $(i = 3, 4, 5)$. The optimum solution is $x^* = (78, 33, 29.995256025682, 45, 36.775812905788)$ where $f(x^*) = -30665.539$. Constraints g_1 y g_6 are active.

5. **g05**
   Minimize:$f(\vec{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$

   subject to:
   $g_1(\vec{x}) = -x_4 + x_3 - 0.55 \leq 0$
   $g_2(\vec{x}) = -x_3 + x_4 - 0.55 \leq 0$
   $h_3(\vec{x}) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$
   $h_4(\vec{x}) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$
   $h_5(\vec{x}) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$

   where $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$, and $-0.55 \leq x_4 \leq 0.55$. The best known solution is $x^* = (679.9453, 1026.067, 0.1188764, -0.3962336)$ where $f(x^*) = 5126.4981$.

6. **g06**
   Minimize: $f(\vec{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$

subject to:
$g_1(\vec{x}) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$
$g_2(\vec{x}) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0$

where $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. The optimum solution is $x^* = (14.095, 0.84296)$ where $f(x^*) = -6961.81388$. Both constraints are active.

7. **g07**
Minimize: $f(\vec{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$
subject to:
$g_1(\vec{x}) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$
$g_2(\vec{x}) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0$
$g_3(\vec{x}) = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$
$g_4(\vec{x}) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$
$g_5(\vec{x}) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0$
$g_6(\vec{x}) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0$
$g_7(\vec{x}) = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$
$g_8(\vec{x}) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$

where $-10 \leq x_i \leq 10$ $(i = 1, \ldots, 10)$. The global optimum is $x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$ where $f(x^*) = 24.3062091$. Constraints g_1, g_2, g_3, g_4, g_5 and g_6 are active.

8. **g08**
Maximize: $f(\vec{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1+x_2)}$
subject to:
$g_1(\vec{x}) = x_1^2 - x_2 + 1 \leq 0$
$g_2(\vec{x}) = 1 - x_1 + (x_2 - 4)^2 \leq 0$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum solution is located at $x^* = (1.2279713, 4.2453733)$ where $f(x^*) = 0.095825$.

9. **g09**
Minimize: $f(\vec{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$
subject to:

$g_1(\vec{x}) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$
$g_2(\vec{x}) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0$
$g_3(\vec{x}) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$

$g_4(\vec{x}) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$

where $-10 \leq x_i \leq 10$ $(i = 1, \ldots, 7)$. The global optimum is $x^* = (2.330499,\ 1.951372,\ -0.4775414,\ 4.365726,\ -0.6244870,\ 1.038131\ ,\ 1.594227)$ where $f(x^*) = 680.6300573$. Two constraints are active (g_1 and g_4).

10. **g10**
    Minimize: $f(\vec{x}) = x_1 + x_2 + x_3$
    subject to: $g_1(\vec{x}) = -1 + 0.0025(x_4 + x_6) \leq 0$
    $g_2(\vec{x}) = -1 + 0.0025(x_5 + x_7 - x_4) \leq 0$
    $g_3(\vec{x}) = -1 + 0.01(x_8 - x_5) \leq 0$
    $g_4(\vec{x}) = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.333 \leq 0$
    $g_5(\vec{x}) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \leq 0$
    $g_6(\vec{x}) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \leq 0$

    where $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$, $(i = 2, 3)$, $10 \leq x_i \leq 1000$, $(i = 4, \ldots, 8)$. The global optimum is: $x^* = (579.19,\ 1360.13,\ 5109.92,\ 182.0174,\ 295.5985,\ 217.9799,\ 286.40,\ 395.5979)$, where $f(x^*) = 7049.25$. g_1, g_2 and g_3 are active.

11. **g11**
    Minimize: $f(\vec{x}) = x_1^2 + (x_2 - 1)^2$
    subject to:
    $h(\vec{x}) = x_2 - x_1^2 = 0$

    where: $-1 \leq x_1 \leq 1$, $-1 \leq x_2 \leq 1$. The optimum solution is $x^* = (\pm 1/\sqrt{2}, 1/2)$ where $f(x^*) = 0.75$.

12. **g12**
    Maximize: $f(\vec{x}) = \frac{100-(x_1-5)^2-(x_2-5)^2-(x_3-5)^2}{100}$
    subject to:
    $g_1(\vec{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$

    where $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$ and $p, q, r = 1, 2, \ldots, 9$. The feasible region of the search space consists of 9^3 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist p, q, r such the above inequality (12) holds. The global optimum is located at $x^* = (5, 5, 5)$ where $f(x^*) = 1$.

13. **g13**
    Minimize: $f(\vec{x}) = e^{x_1x_2x_3x_4x_5}$
    subject to:
    $g_1(\vec{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0$
    $g_2(\vec{x}) = x_2x_3 - 5x_4x_5 = 0$

Problem	n	Function	ρ	LI	NI	LE	NE
g01	13	quadratic	0.0003%	9	0	0	0
g02	20	nonlinear	99.9973%	1	1	0	0
g03	10	nonlinear	0.0026%	0	0	0	1
g04	5	quadratic	27.0079%	0	6	0	0
g05	4	nonlinear	0.0000%	2	0	0	3
g06	2	nonlinear	0.0057%	0	2	0	0
g07	10	quadratic	0.0000%	3	5	0	0
g08	2	nonlinear	0.8581%	0	2	0	0
g09	7	nonlinear	0.5199%	0	4	0	0
g10	8	linear	0.0020%	3	3	0	0
g11	2	quadratic	0.0973%	0	0	0	1
g12	3	quadratic	4.7697%	0	9^3	0	0
g13	5	nonlinear	0.0000%	0	0	1	2

Table 6.1: Values of ρ for the 13 test problems chosen.

$g_3(\vec{x}) = x_1^3 + x_2^3 + 1 = 0$ where $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$. The optimum solution is $x^* = (-1.717143,\ 1.595709,\ 1.827247,\ -0.7636413,\ -0.763645)$ where $f(x^*) = 0.0539498$.

To get an estimate of the difficulty of randomly finding feasible solutions for each of these problems, a ρ metric (as suggested by Michalewicz and Schoenauer [76]) was computed using the following expression: $\rho = |F|/|S|$ where $|F|$ is the number of feasible solutions and $|S|$ is the total number of solutions randomly generated. In this work, $S = 1,000,000$ random solutions.
The different values of ρ for each of the functions chosen are shown in Table 6.1, where n is the number of decision variables, LI is the number of linear inequalities, NI the number of nonlinear inequalities, LE is the number of linear equalities and NE is the number of nonlinear equalities.

6.5 Comparison of Results

We evaluated the performance of our PSO algorithm using the turbulence operator and the constraint-handling mechanism described before. We performed 30 independent runs of our approach for each test function, and we compared our results with respect to three constraint-handling techniques that are representative of the state-of-the-art in the area: Stochastic Ranking (SR) [104], the Homomorphous Maps (HM) [76], and the Adaptive Segregational Constraint Handling Evolutionary Algorithm (ASCHEA) [48]. Our comparison of results is presented in Tables 6.2, 6.3 and 6.4.

Comparing our CPSO approach with respect to the Homomorphous Maps (see Table 6.2), the CPSO technique was able to improve the "best" results in several problems (remarkably in g05, which could not be solved by the homomorphous maps). In the re-

Table 6.2: Comparison of our PSO algorithm with respect to the Homomorphous Maps (HM) [76].

		Best Result		Mean Result		Worst Result	
Problem	**Optimal**	**PSO**	**HM**	**PSO**	**HM**	**PSO**	**HM**
g01	-15.000000	-15.000000	-14.788600	-15.000000	-14.708200	-15.000000	-14.615400
g02	0.803619	0.803432	0.799530	0.790406	0.796710	0.750393	0.791190
g03	1.000000	1.004720	0.999700	1.003814	0.998900	1.002490	0.997800
g04	-30665.539000	-30665.500000	-30664.500000	-30665.500000	-30655.300000	-30665.500000	-30645.900000
g05	5126.498000	5126.640000	NA	5461.081333	NA	6104.750000	NA
g06	-6961.814000	-6961.810000	-6952.100000	-6961.810000	-6342.600000	-6961.810000	-5473.900000
g07	24.306000	24.351100	24.620000	25.355771	24.826000	27.316800	25.069000
g08	0.095825	0.095825	0.095825	0.095825	0.089157	0.095825	0.029144
g09	680.630000	680.638000	680.910000	680.852393	681.160000	681.553000	683.180000
g10	7049.330700	7057.590000	7147.900000	7560.047857	8163.600000	8104.310000	9659.300000
g11	0.750000	0.749999	0.750000	0.750107	0.750000	0.752885	0.750000
g12	1.000000	1.000000	1.000000	1.000000	0.999135	1.000000	0.991950
g13	0.053950	0.068665	NA	1.716426	NA	13.669500	NA

		Best Result		Mean Result		Worst Result	
Problem	**Optimal**	**PSO**	**SR**	**PSO**	**SR**	**PSO**	**SR**
g01	-15.000000	-15.000000	-15.000000	-15.000000	-15.000000	-15.000000	-15.000000
g02	0.803619	0.803432	0.803515	0.790406	0.781975	0.750393	0.726288
g03	1.000000	1.004720	1.000000	1.003814	1.000000	1.002490	1.000000
g04	-30665.539000	-30665.500000	-30665.539000	-30665.500000	-30665.539000	-30665.500000	-30665.539000
g05	5126.498000	5126.640000	5126.497000	5461.081333	5128.881000	6104.750000	5142.472000
g06	-6961.814000	-6961.810000	-6961.814000	-6961.810000	-6875.940000	-6961.810000	-6350.262000
g07	24.306000	24.351100	24.307000	25.355771	24.374000	27.316800	24.642000
g08	0.095825	0.095825	0.095825	0.095825	0.095825	0.095825	0.095825
g09	680.630000	680.638000	680.630000	680.852393	680.656000	681.553000	680.763000
g10	7049.330700	7057.590000	7054.316000	7560.047857	7559.192000	8104.310000	8835.655000
g11	0.750000	0.749999	0.750000	0.750107	0.750000	0.752885	0.750000
g12	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
g13	0.053950	0.068665	0.053957	1.716426	0.057006	13.669500	0.216915

Table 6.3: Comparison of results of our PSO with respect to Stochastic Ranking (SR) [104].

		Best Result		Mean Result		Worst Result	
Problem	**Optimal**	**PSO**	**ASCHEA**	**PSO**	**ASCHEA**	**PSO**	**ASCHEA**
g01	-15.000000	-15.000000	-15.000000	-15.000000	-14.840000	-15.000000	NA
g02	0.803619	0.803432	0.785000	0.790406	0.590000	0.750393	NA
g03	1.000000	1.004720	1.000000	1.003814	0.999890	1.002490	NA
g04	-30665.539000	-30665.500000	30665.500000	-30665.500000	30665.500000	-30665.500000	NA
g05	5126.498000	5126.640000	5126.500000	5461.081333	5141.650000	6104.750000	NA
g06	-6961.814000	-6961.810000	-6961.810000	-6961.810000	-6961.810000	-6961.810000	NA
g07	24.306000	24.351100	24.332300	25.355771	24.660000	27.316800	NA
g08	0.095825	0.095825	0.095825	0.095825	0.095825	0.095825	NA
g09	680.630000	680.638000	680.630000	680.852393	680.641000	681.553000	NA
g10	7049.330700	7057.590000	7061.130000	7560.047857	7193.110000	8104.310000	NA
g11	0.750000	0.749999	0.750000	0.750107	0.750000	0.752885	NA
g12	1.000000	1.000000	NA	1.000000	NA	1.000000	NA
g13	0.053950	0.068665	NA	1.716426	NA	13.669500	NA

Table 6.4: Comparison of results of our PSO with respect to the Adaptive Segregational Constraint Handling Evolutionary Algorithm (ASCHEA) [48].

maining test functions, there is practically a match between the CPSO approach and the homomorphous maps. Regarding average and worst results, the approach proposed here is better than the homomorphous maps in several problems (remarkably, in g05 and g06). No comparisons were made with respect to function g13 because such results were not available for HM.

With respect to Stochastic Ranking (see Table 6.3), CPSO was able to match most of their "best" results. However, stochastic ranking found slightly better results in some problems (remarkably in g10 and g13, which are some of the most difficult test functions from this benchmark). The average and worst results of stochastic ranking are also better than those of the CPSO approach in some problems.

Compared against the Adaptive Segregational Constraint Handling Evolutionary Algorithm (see Table 6.4), CPSO was able to improve the "best" results in several problems (remarkably, in g02 and g10). ASCHEA produced slightly better results only in g03, g05, g07 and g11. We did not compare the worst results because they were not available for ASCHEA. We did not perform comparisons with respect to ASCHEA using functions g12 and g13 for the same reason.

As we can see, our approach showed a very competitive performance with respect to these three state-of-the-art approaches.

The proposed approach can deal with moderately constrained problems (g04), highly constrained problems, problems with low (g06, g08), moderated (g09) and high (g01, g02, g03, g07) dimensionality, with different types of combined constraints (linear, nonlinear, equality and inequality) and with very large (g02), very small (g05 and g13) or even disjoint (g12) feasible regions. Also, the algorithm is able to deal with large search spaces (based on the intervals of the decision variables) and with a very small feasible region (g10). Furthermore, the approach can find the global optimum in problems where such optimum lies on the boundaries of the feasible region (g01, g02, g04, g06, g07, g09).

Note that our approach does not require any parameters. In contrast, the homomorphous maps require an additional parameter (called v) which has to be found empirically [76]. Stochastic ranking requires the definition of a parameter called P_f, whose value has an important impact on the performance of the approach [104]. ASCHEA also requires the definition of several extra parameters, and in its latest version, it uses niching [25], which is a process that also has at least one additional parameter [48].

The computational cost measured in the number of evaluations of the objective function (FFE) performed by our approach is lower than the other techniques with respect to which it was compared. Our approach performed $340,000$ FFE (we used 40 particles running for 8500 generations), the Stochastic Ranking performed $350,000$ FFE, the Homomorphous Maps performed $1,400,000$ FFE, and ASCHEA required $1,500,000$ FFE.

6.6 A Constraint-Handling Mechanism for MOPSO

A relatively simple constraint-handling mechanism for choosing leaders in the particle swarm optimization algorithm has been presented. The proposed approach does not use any special mechanism to deal with constrained search spaces in which the global optimum lies on the boundaries between the feasible and the infeasible regions, despite the fact that such type of problems are the main target of the most competitive constraint-handling techniques

proposed in the specialized literature [104, 48, 76]. Additionally, our CPSO approach does not require any user-defined parameters and it performs less objective function evaluations than any of the other approaches with respect to which it was compared. Despite all of the above reasons, the results obtained by the proposed approach are highly competitive, and in some cases, even improve on the results obtained by much more elaborate approaches such as the homomorphous maps [76] and ASCHEA [48].

Since the aim of this chapter is to introduce a constraint-handling mechanism for our MOPSO, the approach shown in the Algorithm 29 was added into the MOPSO algorithm developed in Chapter 5. In Algorithm 30, the resulting algorithm is shown.

Algorithm 30 CMOPSO

Require: $nsubswarms, nparticles, nconst, dimensions, W, C1$ and $C2$

$\vec{gbest} \leftarrow \vec{x_0}$
$GBEST_{final} \leftarrow \emptyset$
for $s = 0$ to $nsubswarms$ **do**
  $GBEST^s \leftarrow \emptyset$
  for $i = 0$ to $nparticles/nsubswarms$ **do**
    $\vec{pbest}_i^s \leftarrow \vec{x_i^s} \leftarrow initialize_randomly()$
    $fitness_i^s \leftarrow f(\vec{x_i^s})$
    $velocity_{id}^s \leftarrow 0$
    if $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{gbest}))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{gbest}) \cup (\neg\exists\ \vec{y}^* \in GBEST^s \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness}_i^s)))$ **then**
      $GBEST^s \leftarrow GBEST^s \cup \vec{x_i^s}$
    end if
  end for
end for
repeat
  $GBEST_{temp} \leftarrow \emptyset$
  for $s = 0$ to $nsubswarms$ **do**
    for $i = 0$ to $nparticles$ **do**
      if flip(0.5) **then**
        $gbest \leftarrow GBEST^s_{U(0,|GBEST^s|)}$
      else
        $gbest \leftarrow y|y \in GBEST^s \cup \nexists\ z \in GBEST^s|z - x_i^s < y - x_i^s$
      end if
      if x^s is equal $gbest$ **then**
        turbulence (x^s)
      end if
      for $d = 0$ to $ndimensions$ **do**
        $velocity_{id}^s \leftarrow W \times velocity_{id}^s + C_1 \times U(0,1) \times (pbest_{id}^s - x_{id}^s) + C_2 \times U(0,1) \times (gbest - x_{id}^s)$
        $x_{id}^s \leftarrow x_{id}^s + velocity_{id}^s$
      end for
      $fitness_i^s \leftarrow f(\vec{x_i^s})$
      if $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{pbest}_i^s))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{pbest}_i^s) \cup fitness_i^s \sim f(\vec{pbest}_i^s))$ **then**
        $\vec{pbest}_i^s \leftarrow \vec{x_i^s}$
      end if
      if $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{gbest}))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{gbest}) \cup (\neg\exists\ \vec{y}^* \in GBEST^s \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness}_i^s)))$ **then**
        $GBEST^s \leftarrow GBEST^s \cup \vec{x_i^s}$
      end if
    end for
    $GBEST_{final} \leftarrow GBEST_{final} \cup GBEST^s$
  end for
  $GBEST_{final} \leftarrow approach - to - distribute - solutions(GBEST_{final})$
  group $GBEST_{final}$ into $nsubswarms$
  for $s = 0$ to $nsubswarms$ **do**
    $GBEST_s \leftarrow$ randomly select a group formed in last step
  end for
until Termination criterion

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	———-	1.79072	0.123801
NSGA-II	0.00125807	———-	0.0239883
ϵ-MOEA	0.0126029	1.15491	———-

Table 6.5: Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for Kita's test function.

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	———-	0.951662	0.969789
NSGA-II	0.350427	———-	0.74359
ϵ-MOEA	0.449438	0.865169	———-

Table 6.6: Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for Kita's test function.

6.7 CMOPSO's Comparison of Results

In the following examples, the NSGA-II was run using a population size of 40, a crossover rate of 0.8 (uniform crossover was adopted), tournament selection, and a mutation rate of $1/N$, where N = number of variables (real representation was adopted), a distribution index of 15 for real-coded crossover, a distribution index of 20 for real-coded mutation. The ϵ-MOEA was run using a population size of 40, a crossover rate of 0.8 (uniform crossover was adopted), distribution index of 15 for real-coded crossover and distribution index of 20 for real-coded mutation. CMOPSO used 40 particles and a total of 8 swarms.

The total number of fitness function evaluations was set to $3,000$ for all the algorithms compared (75 generations).

6.7.1 Kita's Test Function

Figure 6.2 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in Kita's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the median result with respect to the generational distance metric. Tables 6.5, 6.6 and 6.7, show the comparison of results among the three algorithms considering hyper volume, two set coverage, inverted generational distance, generational distance and success counting metrics. It can be seen that our MOPSO performed best with respect to HV, TSC, IGD and SC.

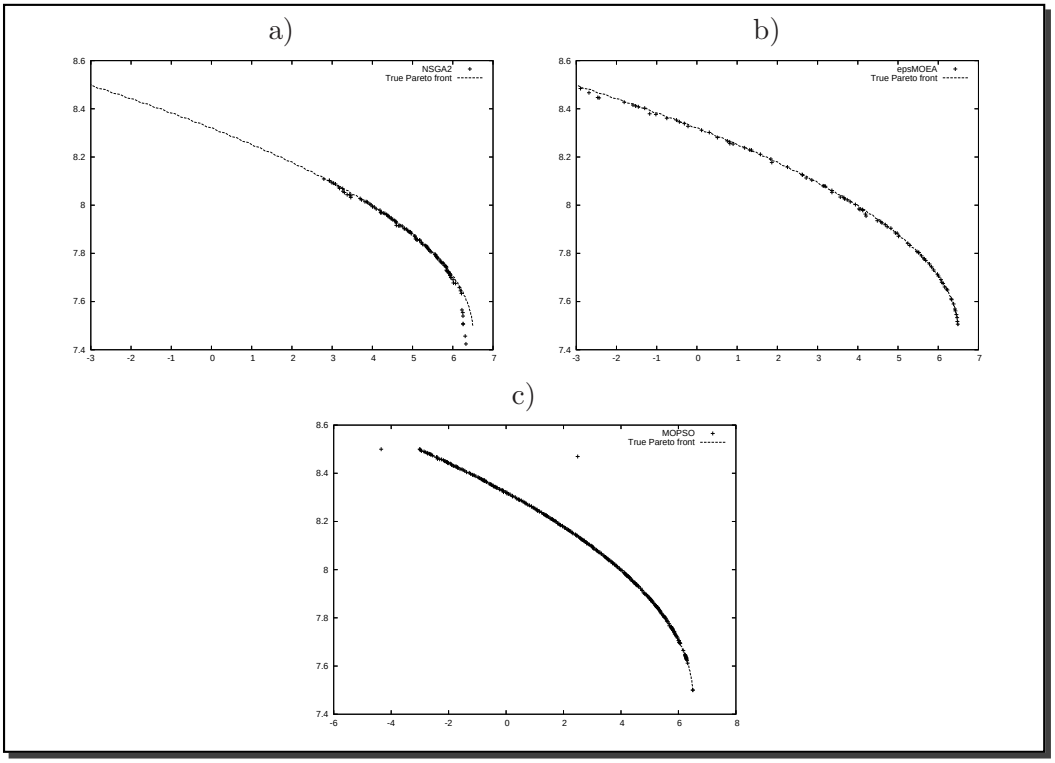

Figure 6.2: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for Kita's test function.

Metric	Statistics	CMOPSO	NSGA-II	ϵ-MOEA
IGD	**Mean**	0.00632531	0.189393	0.0117331
	Best	0.00534285	0.145067	0.00844853
	Worst	0.00816045	0.250597	0.0174223
	St. dev.	0.000754439	0.0393162	0.00332792
	Median	0.00604803	0.179181	0.010696
SC	**Mean**	15.6333	12.4	11.3667
	Best	21	9	8
	Worst	9	18	16
	St. dev.	3.30604	2.37225	3.02271
	Median	15.5	14	16

Table 6.7: Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success Counting(SC) for Kita's test function.

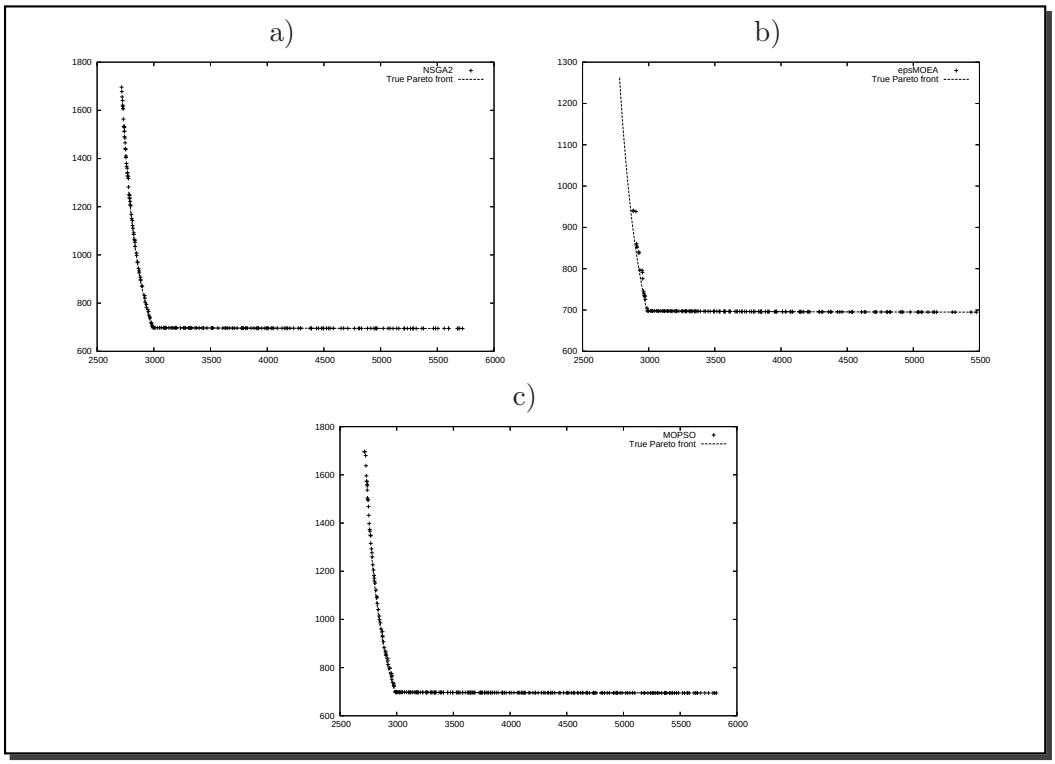

Figure 6.3: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for the Speed Reducer test function.

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	——-	564.319	76326.8
NSGA-II	3593.99	——-	79488.3
ϵ-MOEA	183.015	331.954	——-

Table 6.8: Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for the Speed Reducer test function.

6.7.2 Speed Reducer's Test Function

Figure 6.3 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in the speed reducer test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the median result with respect to the generational distance metric. Tables 6.8, 6.9 and 6.10, show the comparison of results among the three algorithms considering hyper volume, two set coverage, inverted generational distance, generational distance and success counting metrics. In this test function, our MOPSO, could cover best the true Pareto front (it obtained the best result with respect to the TSC metric). However, it can be seen that our MOPSO performed worst with respect to TSC, and SC. With respect to IGD the NSGA-II and ϵ-MOEA performed best, in this metric, MOPSO occupied the second place.

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	———-	0.490196	0.584314
NSGA-II	0.885572	———-	0.81592
ϵ-MOEA	0.883721	0.860465	———-

Table 6.9: Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for the Speed Reducer test function.

Metric	Statistics	CMOPSO	NSGA-II	ϵ-MOEA
IGD	**Mean**	6.69717	29.2915	1.50202
	Best	2.12326	2.60441	1.08238
	Worst	89.5434	100.719	1.83362
	St. dev.	16.0311	39.3283	0.208184
	Median	2.3702	3.06353	1.55295
SC	**Mean**	19.4667	27.9333	34.8333
	Best	29	39	42
	Worst	6	21	0
	St. dev.	5.60008	6.26393	8.28411
	Median	21	25	37.5

Table 6.10: Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success counting(SC) for the Speed Reducer test function.

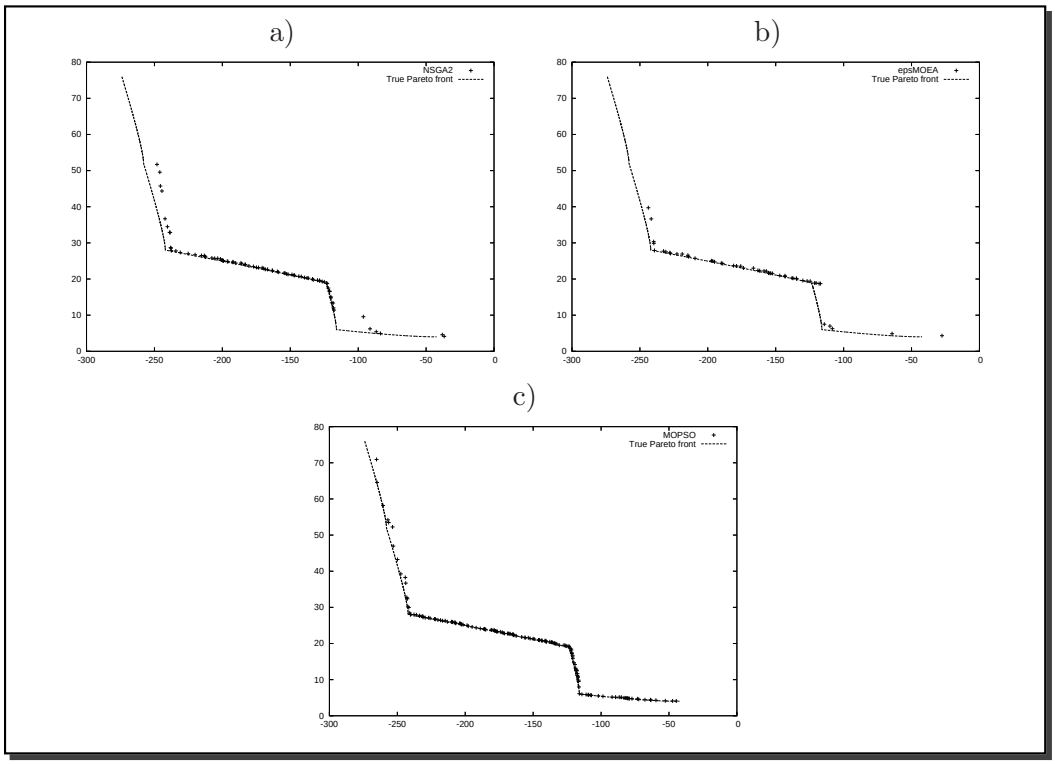

Figure 6.4: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for Osyczka 2's test function.

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	———	528.381	591.783
NSGA-II	5.08483	———	108.539
ϵ-MOEA	2.40862	126.178	———

Table 6.11: Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for Osyczka 2's test function.

6.7.3 Osyczka 2's Test Function

Figure 6.4 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in Osyczka 2's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the median result with respect to the generational distance metric. Tables 6.11, 6.12 and 6.13, show the comparison of results among the three algorithms considering hyper volume, two set coverage, inverted generational distance, generational distance and success counting metrics. It can be seen that our MOPSO performed best with respect to HV, TSC, IGD, and SC.

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	———-	0.823899	0.886792
NSGA-II	0.47619	———-	0.77381
ϵ-MOEA	0.392157	0.627451	———-

Table 6.12: Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for Osyczka 2's test function.

Metric	Statistics	CMOPSO	NSGA-II	ϵ-MOEA
IGD	**Mean**	0.237056	0.468132	0.259819
	Best	0.0784418	0.138079	0.189955
	Worst	1.18076	1.1167	0.318806
	St. dev.	0.223725	0.329838	0.0536878
	Median	0.186781	0.413686	0.265964
SC	**Mean**	0.0333333	0	0
	Best	1	0	0
	Worst	0	0	0
	St. dev.	0.182574	0	0
	Median	0	0	0

Table 6.13: Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success counting(SC) for Osyczka 2's test function.

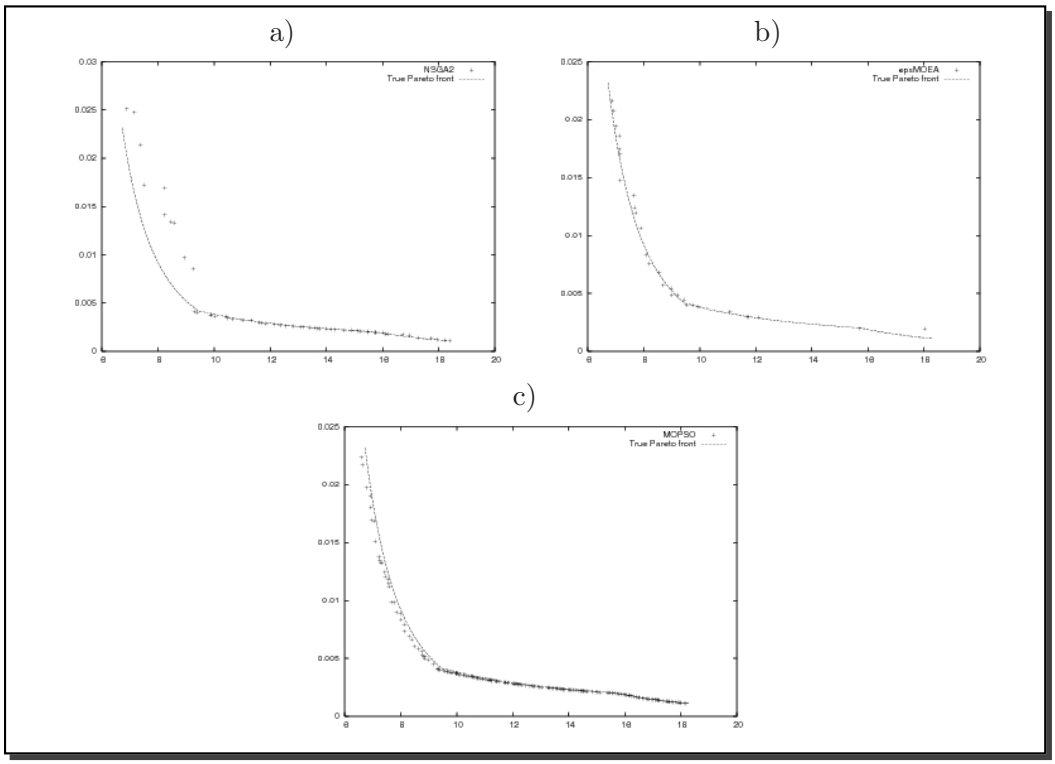

Figure 6.5: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) CMOPSO, for the Welded Beam test function.

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	———-	0.0203806	0.00786607
NSGA-II	2.39609e-05	———-	0.00371345
ϵ-MOEA	3.19206e-05	0.0153286	———-

Table 6.14: Comparison of results of the CMOPSO, NSGA-II, and ϵ-MOEA with respect to Hyper Volume metric for the Welded Beam test function.

6.7.4 Welded Beam's Test Function

Figure 6.5 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in the welded beam function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the median result with respect to the generational distance metric. Tables 6.14, 6.15 and 6.16, show the comparison of results among the three algorithms considering hyper volume, two set coverage, inverted generational distance, generational distance and success counting metrics. It can be seen that our MOPSO performed best with respect to HV, TSC, IGD and SC by far. In figure 6.5 it can be seen that our MOPSO produced solutions that out performed the other approaches.

Algorithm	CMOPSO	NSGA-II	ϵ-MOEA
CMOPSO	———-	0.966667	1
NSGA-II	0.388889	———-	0.814815
ϵ-MOEA	0.214286	0.75	———-

Table 6.15: Comparison of results of the MOPSO, NSGA-II, and ϵ-MOEA with respect to Two Set Coverage metric for the Welded Beam test function.

Metric	Statistics	CMOPSO	NSGA-II	ϵ-MOEA
IGD	**Mean**	0.0243134	0.0383617	0.107524
	Best	0.0090833	0.015111	0.0891251
	Worst	0.0635246	0.05969	0.205429
	Stdev	0.0162971	0.0150994	0.0273268
	Median	0.0136304	0.0362782	0.0917293
SC	**Mean**	15.5333	9.56667	2.4
	Best	31	16	5
	Worst	3	2	0
	Stdev	9.4202	4.77554	2.31338
	Median	20	16	4

Table 6.16: Comparison of results of the CMOPSO, NSGA-II and ϵ-MOEA with respect to inverted generational distance (IGD) and Success counting(SC) for the Welded Beam test function.

6.8 Conclusions

We have presented a relatively simple constraint-handling mechanism for choosing leaders in the particle swarm optimization algorithm. The proposed approach does not use any special mechanism to deal with constrained search spaces in which the global optimum lies on the boundaries between the feasible and the infeasible regions, despite the fact that such type of problems are the main target of the most competitive constraint-handling techniques proposed in the specialized literature [104, 48, 76]. Additionally, our constraint-handling approach does not require any user-defined parameters and it performs less objective evaluations than any of the other approaches with respect to which it was compared (for global optimization). Despite all of the above reasons, the results obtained by our approach are highly competitive, and in some cases, even improve on the results obtained by much more elaborate approaches such as the homomorphous maps [76] and ASCHEA [48]. This constraint-handling mechanism was adapted into our MOPSO. Results indicate that our MOPSO was able to successfully solve several test functions taken from the multiobjective optimization literature.

7

Parameter Control in Multiobjective Particle Swarm Optimization

7.1 Introduction

THE behavior of an EA for a given problem is directed by both the operators (crossover, mutation, etc), and the values selected for the parameters of the algorithm. The issue of setting the values of the parameters plays a key role on the performance of an EA. Tuning well these parameters is a hard problem, since they can usually take several values, and therefore, the number of possible combinations is usually very high.

So, we can say that there is an additional optimization problem to solve: parameter tuning. It is only natural to expect that an evolutionary or adaptive scheme is adopted to solve this additional optimization problem (this is called self-adaptation in evolutionary computation). However, parameter control is one of the research topics that has been only scarcely covered in the current literature on evolutionary multiobjective optimization [19].

The parameter adaptation problem is an important issue since evolutionary multiobjective optimization techniques usually require more parameters than a traditional evolutionary algorithm (e.g., a niche radius or sharing threshold). In this chapter we focus on the use of adaptive processes to adjust parameters. First, we begin by giving a short explanation about adaptation, and parameter adaptation. Next, we revise the previous related work. Then, we perform an analysis of the parameters of our MOPSO. After that, we propose a parameter adaptation scheme for our MOPSO. We validate this proposal by comparing our adaptation scheme with respect to a fixed selection of the parameters (derived from the parameter analysis). Then, we compare our results with respect to those obtained by NSGA-II and ϵ-MOEA. Finally, we validate our proposal using several test functions and metrics, and we provide some conclusions.

7.2 Tuning or Adapting Parameters

Parameter tuning refers to the selection of good values for the parameters before the run of the algorithm (these values remain fixed along the run of the algorithm). This is a commonly used approach. However to select the best parameters' settings for several problems, it is necessary to perform experiments with a wide variety of values for such parameters. This methodology is very expensive (computationally speaking), since it is a very time-consuming activity. Furthermore, a set of parameters that behave well on some problems can behave very badly on others.

Adaptation is a process usually invoked to improve the overall performance of an EA.

In general, there are two types of adaptation [41]:

- Adaptation to problems
- Adaptation to evolutionary processes

The first issue refers to performing a modification of one or more components of an EA (i.e., variation operators, selection or representation), in order to solve faster and with more accuracy a given problem. The second issue refers to modifying the values of the parameters with the aim of avoiding any fine tuning from the user. "Adaptation to problems" refers to an ad-hoc modification of an EA to a particular problem, and "adaptation to evolutionary processes" refers to modifications to the evolutionary process itself with the aim to perform better (i.e. to the variation operators). The second issue has been studied extensively in the literature.

Adaptation to evolutionary processes can be divided into the following classes [49]:

- Adaptive parameters settings
- Adaptive genetic operators
- Adaptive selection
- Adaptive representation
- Adaptive fitness function

Parameter adaptation has been the most studied class in the last few years, since Davis [22] and Grefenstette [46] found that the mutation rate, crossover rate and population size have an important role on the performance of an evolutionary algorithm (specifically on a genetic algorithm).

7.3 Parameter Adaptation

The key factor to increase the performance of an EA is to choose an appropriate trade-off between exploitation and exploration. However, if we want to perform a good balance of them on a particular EA, it is necessary to use a set-and-test approach. Nevertheless, the values obtained are usually (in most EAs) dependant of the specific problem at hand. Therefore, if we want to increase the performance of an EA on a wide variety of problems

we have to modify the values of the exploitation and exploration by using a deterministic rule, taking feedback information from the current state of the search (on-line adaptation mechanism), or employing a self-adaptive mechanism [51, 41, 49]:

- Deterministic adaptation refers to a modification of a parameter using a deterministic rule (i.e. the mutation rate is decreased gradually over time).

  Several researchers have used deterministic adaptation in the past [34, 50, 52]. However, they have used it only on parameters whose behavior is well known.

- Online-adaptation refers to a modification performed on the value of a parameter whose direction and magnitude is determined by some information taken from the evolutionary process.

  An example of this type of adaptation is the $\frac{1}{5}$ success rule proposed by Rechenberg to adapt the standard deviation in an evolution strategy [101].

- Self-adaptation refers to a modification of a parameter directed by the evolutionary process. In this case, the parameters are encoded into the chromosome string, and the variation operators are applied to them.

  This type of adaptation can be used to control the application of different variation operators.

7.4 Related Work

There have been very few attempts in the literature to produce an evolutionary multiobjective optimization technique that adapts its parameters during the evolutionary process and that, therefore, does not require any fine-tuning from the user. One of the earliest attempts to incorporate self-adaptation mechanisms in evolutionary multiobjective optimization was Kursawe's proposal of providing individuals with a set of step sizes for each objective function such that his multiobjective evolution strategy could deal with a dynamic environment [77]. Laumanns et al. [78] showed that a standard self-adaptive evolution strategy had problems to converge to the true Pareto set of a multiobjective optimization problem and proposed alternative self-adaptation mechanisms that, however, were applied only to an aggregating fitness function. Tan et al. [113] proposed the incrementing multiobjective evolutionary algorithm (IMOEA), which uses a dynamic population size that adapts based on the tradeoffs produced so far and the desired population distribution density. The IMOEA relies on a measure of convergence based on population domination and progress ratio [117]. The IMOEA also uses dynamic niches (i.e., no sharing factor needs to be defined). Another interesting proposal is the idea of Büche et al. [8] of using self-organizing maps of Kohonen [72] to adapt the mutation step size of an evolutionary multiobjective optimization algorithm. The authors also define a recombination operator using self-organizing maps (something similar to intermediate recombination). Abbass [1] recently proposed a differential evolution algorithm used to solve multiobjective problems that self-adapts its crossover and mutation rates. Zhu and Leung [129] proposed an asynchronous self-adjustable island genetic algorithm in which certain information about the current search status of each island in a parallel evolutionary algorithm is used to focus the search effort into non-overlapping regions.

7.5 MOPSO: Parameter's Analysis

The PSO has three parameters that play a key role in the algorithm's behavior (see Chapter 3):

1. W: velocity inertia
2. C1: cognitive component
3. C2: Social Component

The fact that a MOEA converges to a set of solutions rather than to a single value, makes it difficult to perform an statistical analysis such as the analysis of variance, which can determine how sensitive is an algorithm to its parameters. Nevertheless, we will perform a very thorough analysis of parameters (similar to an analysis of variance), with the aim of finding the best possible parameter configuration for our approach (considering the set of test functions adopted).

It is worth indicating that the parameters settings that have been previously proposed (see for example [65]) for the original (unconstrained single-objective) PSO, do not provide a good performance in the context of multiobjective optimization and therefore the motivation to perform the thorough analysis reported in this chapter.

In order to analyze the impact of the parameters on our proposed approach, we considered several configurations, and performed a comprehensive number of runs. The configurations adopted are:

$$\begin{aligned} W &= \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ C1 &= \{1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0\} \\ C2 &= \{1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0\} \end{aligned} \tag{7.1}$$

For all our experiments, we adopted 40 particles. However, we analyzed three different performance scenarios:

1. **Experiment 1:** Use of a randomly generated initial population and a low number of fitness function evaluations (we used $Gmax = 25$, which gives us a total of 1000 fitness function evaluations).
2. **Experiment 2:** Use a good approximation of the Pareto front in the initial population. This approximation, although hasn't converged to the true Pareto front. This approximation was selected by hand based on a visual analysis of different runs. In all the experiments performed in this case, the same approximation was fed to the algorithm in its initial generation. In this case, we only performed 600 fitness function evaluation ($Gmax = 15$), since we were interested in analyzing the capability (or possible difficulties) of our algorithm to reach the true Pareto front of a problem once a good (and sufficiently close) approximation has been produced.
3. **Experiment 3:** Use of a large number of fitness function evaluations ($Gmax = 250$, which gives a total of $10,000$ fitness function evaluations) in order to assess the performance of our approach in the long term.

We adopted eleven test functions for **Experiment 1** and **Experiment 2**. Due to the high CPU time required by each run, we only adopted six test functions for **Experiment 3**.

Since we needed to assess performance in each case, a metric had to be adopted. We chose inverted generational distance (see Chapter 2) because it can measure both closeness to the true Pareto front and spread of solutions.

An obvious problem with so many experiments was how to present the results in a compact form. For that sake, we adopted a set of squares (called "mosaics"), such that each of them has a color that corresponds to the mean value of the inverted generational distance over 30 independent runs, produced from one combination of W,C1,C2 (all the possible combinations were adopted, considering the sets of possible values previously defined for these three parameters).

The mean results are normalized between zero and 255 (where zero is the best possible value and 255 is the worst). So, a mosaic that is completely black represents the best possible mean values, and a mosaic completely white represents the worst possible mean values. The results of each of the three above experiments are briefly discussed next, to avoid having a chapter excessively long, the mosaics have been moved to Appendix B.

7.5.1 Experiment 1

Figures B.1, B.2, B.3, B.4, B.5, B.6, B.7, B.8, B.9, B.10, B.11 show the mosaic for the test functions of Kursawe, Deb1, Deb2, ZDT1, ZDT2, ZDT3, ZDT6, Kita, Welded Beam, Ozyczka2 and Speed reducer respectively (see Section 2.5, for each test function description).

Summarizing the graphical results, we can say the following:

- As expected (as a consequence of the No Free Lunch Theorem [124]), no single set of parameters was found to be the best over all the test functions adopted.
- However, the region surrounding C1=1.4, C2=1.4 and W=0.2 (this includes the following: C1 can vary from 1.2 to 2.0, C2 can vary from 1.2 to 2.0 and W can vary from 0.1 to 0.4) shows a good performance in most cases.
- It is worth noticing that Figures B.4, B.5, B.6 and B.7 show an behavior that is the opposite from the one observed in the other figures. The reason is that the test functions illustrated in these figures have their true Pareto front located in the lower bound of all the decision variable. Thus, any decision variable values above the lower bound cause a disruptive behavior in the algorithm.

7.5.2 Experiment 2

Figures B.12, B.13, B.14, B.15, B.16, B.17, B.18, B.19, B.20, B.21 and B.22 show the mosaic for the test functions of Kursawe, Deb, Deb2, ZDT1, ZDT2, ZDT3, ZDT6, Kita, Welded Beam, Ozyczka2 and Speed reducer respectively (see Section 2.5, for each test function description).

Our conclusions from the second experiment are the following:

- Again, no single combination of parameters produced the best possible results for all the test functions adopted.
- However, we found a narrow range of values for which reasonably good results are obtained: C1 and C2 should be within 1.0 and 1.2, and W within 0.1 and 0.2 (see Figures B.12, B.13, B.14, B.20 and B.21)
- There is also (as in the previous experiment) a set of test functios for which the results are inconclusive (see Figures B.15, B.16, B.17, B.18, B.19 and B.22).

7.5.3 Experiment 3

Figures B.23, B.24, B.25, B.26, B.27 and B.28 show the mosaic for the test functions of Kursawe, Deb2, ZDT3, Kita, Welded Beam and Ozyczka2 respectively (see Section 2.5, for each test function description for all the test functions adopted).

The main conclusions derived from the third experiment are the following:

- Practically all combinations of parameters produce a good performance, which indicates that our approach is able to converge to the true Pareto front of all the problems if given a sufficiently long time.
- Although this experiment indicates in general, that our approach has no real sensitivity to the values of C1 and C2 when performing a high number of fitness function evaluations, we found that our approach is sensitive to values of W above 0.8.

7.5.4 Conclusions from the Experiments

After combining the results from the three experiments, we concluded the following:

- The best range for C1 and C2 is from 1.2 to 2.0. From within this range, we can say that C1=C2=1.4 provides the best overall performance.
- The best values for W are those less or equal to 0.5.
- Note, however, that the above values don't produce the best possible performance in all cases. This is precisely what motivated us to propose a mechanism to self-adapt the parameters of our approach so that it automatically adjusts the parameters according to the characteristics of the search space being explored.

7.6 Self-Adaptation Mechanism

The results obtained from the experiments reported in the previous section led us to propose a self-adaptation mechanism which is described in this section.

We proposed the use of a traditional proportional selection mechanism to select the most appropriate combination of parameters values to be adopted. We adopted roulette-wheel selection [43] for that sake. Rather than computing fitness for each combination of values, we count the number of non-dominated solutions generated by each combination of values.

Let's assume that the parameter W can take 3 possible values: 0, 0.5, 1.0. So, at the beginning of the search, each value has a 33% probability of occurring (see Figure 7.1 (a)). At generation zero, each possible value has a "fitness" of one. Now, let's assume that after one generation, W=0.5 generated 2 and W=1 did not contribute with any new non-dominated solution. Thus, we reward the "fitness" of W=0.5 by increasing its value in 0.4 (we increase fitness in 0.1 for each new non-dominated solution produced). So, W=0.5 now has a fitness of 1.4. Analogously, W=0 has a fitness of 1.2 and W=1 remains with a fitness of 1.0. The new share in the roulette wheel for each value is shown in Figure 7.1 (b). Since the total fitness is now 3.6 (1.4+1.2+1.0), the share of each value is: W=0.5 (1.4/3.6), W=0 (1.2/3.6) and W=1 (1.0/3.6).

After generation two, W=0.5 generated one new non-dominated solution. the same happened with W=1.0 and W=0 didn't produce any new non-dominated solutions. So, W=0.5 now has a fitness of 1.5, W=1.0 has a fitness of 1.1 and W=0 remains with a fitness of 1.2. Thus, the total fitness is now 3.8, and the share of each value is: W=0.5 has 1.5/3.8, W=1 has 1.1/3.8 and W=0 1.2/3.8 (see Figure 7.1 (c)).

To validate our proposal, we compare it against two parameter selection proposals: a) deterministic selection and b) random selection.

Deterministic selection : $C1$, $C2$ and W are deterministically set to 1.4, 1.4 and 0.2, respectively (these values are the center of the region which performed best from the experiments reported in the previous section).

Random selection : $C1$, $C2$ and W pick their values randomly from an interval from 1.2 to 2 for $C1$ and $C2$ and from the range from 0.0 to 0.4 for W (this is the range of values which performed best in the experiments reported in the previous section).

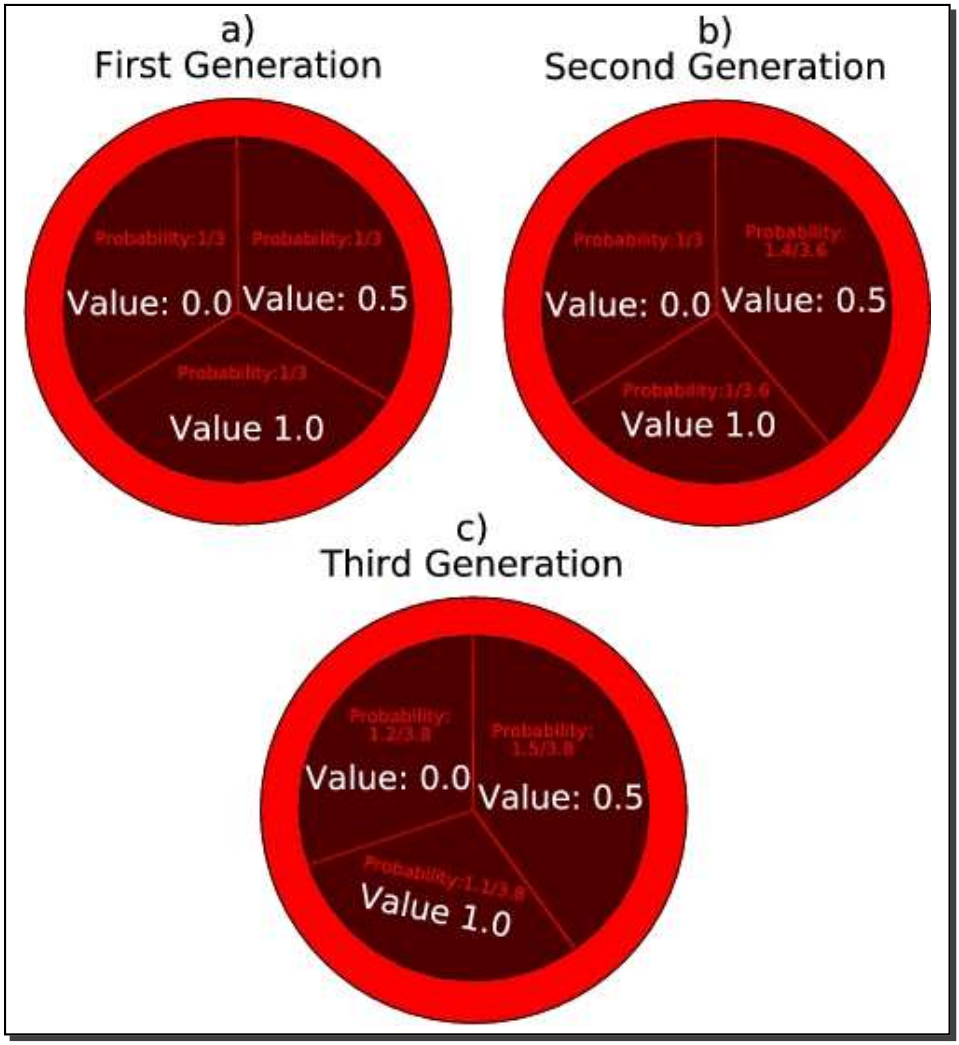

Figure 7.1: Roulette Wheel Selection example at the a) first, b) second and c) third generation.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.000325975	0.000322839	0.00100099	0.000343704
Best	0.000205887	0.000219512	0.000534565	0.000246475
Worst	0.0005185	0.000559256	0.00184197	0.000503136
St. dev.	8.09051E-05	7.93461E-05	0.000369327	6.84881E-05
Median	0.000308359	0.000311513	0.000950674	0.000332008

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.000276673	0.000256187	0.000395132	0.000343704
Best	0.000210781	0.000200151	0.000230807	0.000246475
Worst	0.000427433	0.000403939	0.000578551	0.000503136
St. dev.	5.64173E-05	3.64836E-05	.46397E-05	6.84881E-05
Median	0.000260992	0.000255322	0.000396123	0.000332008

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.000244526	0.000246204	0.00023809	0.000250122
Best	0.00020898	0.000214326	0.000161674	0.000189423
Worst	0.000260916	0.000270411	0.000294859	0.000263756
St. dev.	1.12059E-05	1.24811E-05	3.2749E-05	1.37935E-05
Median	0.000245691	0.000248583	0.000240319	0.000251664

Table 7.1: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Deb1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.1 Deb1's Test Function

Figures C.1, C.2 and C.3 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self-adaptation and d) self-adaptation using half of the parameters' range in Deb1's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.1 and 7.2 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. We can see that in this test function, the fixed and random methodologies to adapt the parameters were the approaches that performed best.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	22.4667	20.8667	12.4333	19.7333
Best	30	29	20	25
Worst	15	13	5	12
St. dev.	4.2486	4.31304	3.58813	2.82761
Median	21	22	15	19

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	26.9667	27.9667	18.8	26.9333
Best	34	36	27	34
Worst	14	20	13	20
St. dev.	4.57492	4.36667	3.12278	3.37264
Median	28	27	19	26.5

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	37.6667	37.6667	29.7333	37.3667
Best	41	41	36	41
Worst	33	31	24	31
St. dev.	2.17086	2.35377	3.37264	2.44221
Median	38	38	30	37

Table 7.2: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Deb1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.281854	0.205165	0.137337	0.209693
Best	0.0154214	0.00119245	0.00146099	0.00149948
Worst	0.390221	0.414155	0.409451	0.582554
St. dev.	0.105455	0.146787	0.1553	0.161553
Median	0.304469	0.302041	0.0383962	0.301602

100 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.085119	0.152535	0.0291952	0.133702
Best	0.00108132	0.00120821	0.00112473	0.00111747
Worst	0.371893	0.415602	0.342749	0.358148
St. dev.	0.129034	0.162967	0.0804286	0.145663
Median	0.0201443	0.0316255	0.00166208	0.0317042

250 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.0118809	0.0114155	0.00293867	0.0249622
Best	0.000948934	0.000979696	0.00109274	0.00102406
Worst	0.304198	0.304024	0.0347976	0.354865
St. dev.	0.0552553	0.05527	0.00670915	0.0830927
Median	0.00118685	0.00120164	0.00128125	0.00127768

Table 7.3: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Deb2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.2 Deb2's Test Function

Figures C.4, C.5 and C.6 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in Deb2's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Table 7.3 shows the comparison of results among the four algorithms considering the generational distance metric. In this test function, fixed and self-adaptive were the approaches which performed best.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.00798041	0.00887023	0.0287635	0.0100363
Best	0.00515281	0.00646239	0.0142215	0.00704959
Worst	0.010944	0.0140615	0.0704334	0.0137772
St. dev.	0.00165173	0.00174998	0.0120881	0.00192299
Median	0.00765617	0.00829771	0.0265748	0.00973348

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.0060196	0.00622204	0.0113361	0.00661385
Best	0.00461587	0.00510085	0.00801492	0.00520072
Worst	0.00722763	0.00910346	0.0162751	0.00849508
St. dev.	0.00070381	0.000901342	0.00233548	0.000856513
Median	0.006001	0.00604657	0.0110073	0.00645687

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.00585674	0.00597782	0.00661119	0.00568906
Best	0.00503107	0.00508054	0.00525094	0.00516091
Worst	0.00757255	0.0103294	0.00834815	0.00705431
St. dev.	0.000542949	0.00106411	0.000792981	0.000414757
Median	0.00574721	0.00568223	0.00652533	0.00567236

Table 7.4: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Kursawe's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.3 Kursawe's Test Function

Figures C.7, C.8 and C.9 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self-adaptation and d) self-adaptation using half of the parameters' range in Kursawe's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.4 and 7.5 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. We can see that in this test function, the fixed and self-adaptive methodologies were the best performers. For 50 and 100 iterations, the fixed methodology perform best. However, for 250, the self-adaptive (using half of the whole range) approach outperformed the other methodologies.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	15.8333	12	2.43333	9.56667
Best	24	19	7	18
Worst	9	7	0	4
St. dev.	3.78822	3.55256	2.02882	3.74795
Median	16.5	15	2	4.5

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	23.8	22.2	7.93333	19.3667
Best	33	30	15	29
Worst	18	13	1	12
St. dev.	3.37741	4.78071	3.21562	4.24657
Median	24.5	22	5	19

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	30.6	30.7667	21.0333	30.3667
Best	37	36	26	36
Worst	22	24	11	22
St. dev.	3.44013	2.82456	3.90829	3.1237
Median	31	30	21	31

Table 7.5: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Kursawe's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.0879494	0.07316	0.125552	0.0860936
Best	0.00482195	0.00440464	0.00487845	0.00438719
Worst	0.753995	0.351501	0.815724	0.894258
St. dev.	0.169574	0.102575	0.168681	0.198588
Median	0.0168816	0.0224311	0.0672385	0.025054

100 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.0909219	0.084916	0.113282	0.0743991
Best	0.00397045	0.00427523	0.00582636	0.00388963
Worst	0.989896	1.33762	1.28782	0.570475
St. dev.	0.214879	0.239521	0.236216	0.136491
Median	0.0174579	0.0280449	0.0310777	0.0202444

250 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.045679	0.0713936	0.0828661	0.0478608
Best	0.00475388	0.00409998	0.00461058	0.00429014
Worst	0.444079	0.69912	0.734926	0.76521
St. dev.	0.0931111	0.17094	0.154571	0.13693
Median	0.00917924	0.0130802	0.0343496	0.0170105

Table 7.6: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Kita's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.4 Kita's Test Function

Figures C.10, C.11 and C.12 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in Kita's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.6 and 7.7 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. We can see that in this test function, the fixed and self-adaptive methodologies were the best performers. For 50 and 100 iterations, the fixed methodology performed best. However, for 250, the self-adaptive (using half of the whole range) approach outperformed the other methodologies.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	10.0667	9.5	4.13333	8.26667
Best	18	18	8	15
Worst	2	4	1	3
St. dev.	4.1683	3.20291	2.22421	2.89986
Median	16	6	4	5

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	13.9	12.5333	7.8	13.3333
Best	23	19	13	20
Worst	3	3	3	6
St. dev.	4.59648	3.27723	2.8211	3.62304
Median	16	13	5.5	14

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	19.7667	19.1	11.6667	18.5667
Best	30	30	19	25
Worst	8	5	6	6
St. dev.	5.30896	5.70753	3.39709	4.09106
Median	22	21	13.5	19

Table 7.7: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Kita's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations				
	Approaches			
Statistics	**Fixed**	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.0366011	0.00314461	0.000917618	0.00139146
Best	0.0033862	0.0016411	0.000654515	0.000949446
Worst	0.128655	0.0153514	0.00129093	0.00194027
St. dev.	0.031028	0.00286637	0.000158416	0.000273769
Median	0.035668	0.00220733	0.000888994	0.00139852

100 iterations				
	Approaches			
Statistics	**Fixed**	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.00248134	0.00111711	0.000713258	0.000900094
Best	0.00104509	0.000877039	0.00055484	0.000718813
Worst	0.00959323	0.00155976	0.000837561	0.00112768
St. dev.	0.00190083	0.000188505	5.64943E-05	9.28948E-05
Median	0.00192676	0.00107593	0.000713413	0.000889186

250 iterations				
	Approaches			
Statistics	**Fixed**	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	0.000885911	0.000722293	0.000669619	0.000690558
Best	0.000690418	0.000606415	0.000627206	0.000660934
Worst	0.00145514	0.000882852	0.00069318	0.000746049
St. dev.	0.000167184	5.18389E-05	1.52374E-05	2.28605E-05
Median	0.000833629	0.000720251	0.000670696	0.000687627

Table 7.8: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.5 ZDT1's Test Function

Figures C.13, C.14 and C.15 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in ZDT1's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.8 and 7.9 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. We can see that in this test function, the self-adaptive methodology was the best performer.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	1.4	5.3	19.0667	9.9
Best	9	13	31	20
Worst	0	0	10	6
St. dev.	2.37225	3.09783	4.6752	3.4074
Median	0	4	18.5	6

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	5.86667	9.9	25.6	16.5
Best	19	17	32	22
Worst	0	4	17	6
St. dev.	4.09148	3.67048	4.07346	4.19153
Median	4	10.5	25	18

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	13.1667	23.2333	34.7	26.6667
Best	23	32	40	33
Worst	6	11	28	14
St. dev.	4.69103	5.10364	2.94958	4.06273
Median	14.5	23.5	35	27.5

Table 7.9: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT1's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.00129201	0.00116102	0.00060177	0.00105732
Best	0	0.000221067	0.000486076	0.000214616
Worst	0.00636153	0.00333243	0.0011321	0.00398105
St. dev.	0.00168527	0.000899109	0.000125183	0.000770421
Median	0.00028753	0.00101155	0.000566384	0.000727372

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Best	0.000208664	0.000252649	0.000513262	0.000518098
Worst	0.0106404	0.00124342	0.000695098	0.000955088
Mean	0.000998943	0.0006525	0.000554823	0.000623259
St. dev.	0.00190153	0.000237816	3.61986E-05	0.000118351
Median	0.000544656	0.000600113	0.000545395	0.000564439

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Best	0.000499733	0.000525243	0.000534474	0.00053733
Worst	0.000901211	0.000673453	0.000560556	0.000595716
Mean	0.000613502	0.000571093	0.000545045	0.000556936
St. dev.	8.77564E-05	3.32668E-05	6.45714E-06	1.76145E-05
Median	0.000580094	0.000559791	0.000545105	0.000551748

Table 7.10: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.6 ZDT2's Test Function

Figures C.16, C.17 and C.18 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in ZDT2's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.10 and 7.11 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. In this test function, the approach that behave best was the Self-adaptive, since it outperformed the others in most of the statistics.

50 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	18.6333	17.9	32.8333	23.7
Best	underline40	underline40	40	underline40
Worst	1	1	14	4
St. dev.	16.7836	14.8657	8.71417	13.9189
Median	33	36	35.5	37

100 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	22.9667	23.1667	36.1	30.9667
Best	40	40	40	40
Worst	1	5	17	11
St. dev.	14.0135	11.0643	5.3842	10.427
Median	36	29.5	37.5	35.5

250 iterations

Statistics	Approaches			
	Fixed	**Random**	**Self-adaptive**	**Self-adaptive half of the range**
Mean	23.2	32.0667	39.1667	35.4
Best	38	underline40	40	40
Worst	7	18	36	25
St. dev.	8.3806	5.66863	1.93129	5.42408
Median	24	32	40	37.5

Table 7.11: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.0469277	0.00337089	0.00130207	0.00283484
Best	0.00128216	0.000845646	0.000961234	0.000989847
Worst	0.159038	0.0136354	0.00257293	0.0236592
St. dev.	0.0437315	0.00263124	0.00038585	0.00404143
Median	0.042308	0.00260087	0.00116183	0.00192842

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.00838771	0.00160257	0.00111052	0.00137749
Best	0.000932991	0.00104869	0.000796095	0.00100488
Worst	0.0443346	0.00329232	0.0014914	0.00179113
St. dev.	0.0102217	0.000470434	0.000146541	0.000190331
Median	0.00320439	0.0014143	0.00111442	0.00140797

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.00118112	0.00105378	0.000995126	0.00098538
Best	0.000906703	0.000903995	0.000836458	0.000862196
Worst	0.00148058	0.00120019	0.00109687	0.00107781
St. dev.	0.000166797	6.71042E-05	4.80638E-05	4.96751E-05
Median	0.00117058	0.0010517	0.000986394	0.000987349

Table 7.12: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT3's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.7 ZDT3's Test Function

Figures C.19, C.20 and C.21 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in ZDT3's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.12 and 7.13 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. In this test function, the self-adaptive methodology was the approach that produced the best results.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.5	2.26667	7.26667	3.1
Best	4	6	13	8
Worst	0	1	1	0
St. dev.	1.00858	1.50707	2.98194	1.74889
Median	0	1.5	6	3

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	1.86667	4.03333	13.8667	6.03333
Best	10	11	26	13
Worst	0	1	5	2
St. dev.	2.56949	2.45628	5.62466	3.0341
Median	1	3	17.5	4.5

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	10.5	14.1333	20.8667	17.8667
Best	22	20	26	27
Worst	2	6	16	11
St. dev.	4.95323	3.45147	2.58288	3.30864
Median	21	14	21	18

Table 7.13: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT3's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Fixed	Random	Self-adaptive	Self-adaptive half of the range
	Approaches			
Mean	0.197208	0.157905	0.15285	0.218335
Best	0.000217578	0.000200907	0.000164541	0.000174668
Worst	0.671328	0.684974	1.15015	1.08311
St. dev.	0.170137	0.159985	0.284782	0.260298
Median	0.132875	0.119427	0.000227409	0.169256

100 iterations

Statistics	Fixed	Random	Self-adaptive	Self-adaptive half of the range
	Approaches			
Mean	0.118331	0.180459	0.154155	0.191904
Best	0.000150442	0.000192441	0.000162983	0.000170195
Worst	0.360862	0.650162	1.23347	0.731475
St. dev.	0.10547	0.171686	0.292953	0.205481
Median	0.0804339	0.172421	0.000217733	0.156286

250 iterations

Statistics	Fixed	Random	Self-adaptive	Self-adaptive half of the range
	Approaches			
Mean	0.0468028	0.097265	0.152427	0.103105
Best	0.000195265	0.000183633	0.000171729	0.000178926
Worst	0.272685	0.526762	0.89369	0.640128
St. dev.	0.0678927	0.139964	0.258548	0.167601
Median	0.0127627	0.0190864	0.000216973	0.000224342

Table 7.14: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for ZDT6's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.8 ZDT6's Test Function

Figures C.22, C.23 and C.24 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in ZDT6's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.14 and 7.15 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. In this test function, the best behavior was shared by the fixed and the self-adaptive approaches.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	23.2333	24.6	26.5667	24.2667
Best	36	36	36	38
Worst	12	10	9	10
St. dev.	6.07246	7.41201	8.23652	7.75012
Median	22	26.5	30.5	25

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	26.4333	24.5333	26.4	24.3333
Best	35	33	37	34
Worst	13	11	10	8
St. dev.	5.69745	6.60059	7.88101	7.08244
Median	26.5	24.5	30	27

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	28.8	29.1667	26.5667	27.2
Best	36	36	36	33
Worst	13	16	9	10
St. dev.	5.94457	5.47145	8.23652	6.01951
Median	31	30	30.5	30.5

Table 7.15: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for ZDT6's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.194347	0.085628	0.0209355	0.0719164
Best	0.0111299	0.00178059	0.00162807	0.00143289
Worst	0.464535	0.449933	0.0999722	0.399702
St. dev.	0.175667	0.131092	0.0274426	0.117621
Median	0.101435	0.0182097	0.00808533	0.0151489

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.133814	0.0631828	0.025962	0.0668184
Best	0.0011115	0.000376051	0.00052514	0.000563104
Worst	0.383953	0.313176	0.341498	0.392752
St. dev.	0.156396	0.100886	0.0659602	0.123076
Median	0.0428005	0.0135748	0.00301419	0.00514221

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.0680538	0.0825331	0.0150598	0.0454222
Best	0.000447409	0.000437696	0.00036197	0.000488802
Worst	0.379767	0.384567	0.232387	0.348925
St. dev.	0.122755	0.143884	0.0453287	0.102489
Median	0.0088608	0.00437517	0.00131789	0.00404749

Table 7.16: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for the Welded Beam test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.9 Welded Beam Test Function

Figures C.25, C.26 and C.27 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in the Welded Beam test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.16 and 7.17 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. Both self-adaptive approaches were the methodologies which behaved best in this test function.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	1.56667	1.86667	1.6	1.76667
Best	2	5	3	3
Worst	1	1	1	1
St. dev.	0.504007	0.937102	0.723974	0.678911
Median	2	2	1	2

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	2.06667	2.43333	2.2	2.46667
Best	4	6	4	5
Worst	1	1	1	1
St. dev.	0.52083	1.04	0.886683	0.860366
Median	2	2	2	2

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	2.46667	2.53333	3.26667	2.96667
Best	6	5	7	7
Worst	2	2	2	2
St. dev.	0.937102	0.899553	1.41259	1.29943
Median	2	2	3	3

Table 7.17: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for the Welded Beam test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Fixed	Random	Self-adaptive	Self-adaptive half of the range
	Approaches			
Mean	2.03861	2.13542	2.11281	1.54764
Best	0.642933	0.548188	0.193383	0.196416
Worst	4.92499	5.74446	5.55049	4.94761
St. dev.	1.23967	1.25894	1.90638	1.36253
Median	1.55246	1.72205	0.984932	0.960847

100 iterations

Statistics	Fixed	Random	Self-adaptive	Self-adaptive half of the range
	Approaches			
Mean	1.17476	0.819187	0.840237	0.916743
Best	0.291657	0.123535	0.116173	0.178783
Worst	4.08389	2.46626	3.55097	3.40832
St. dev.	0.888326	0.504245	0.855718	0.896507
Median	0.918393	0.7161	0.556836	0.63004

250 iterations

Statistics	Fixed	Random	Self-adaptive	Self-adaptive half of the range
	Approaches			
Mean	0.736474	0.945802	0.752102	0.723455
Best	0.133017	0.150762	0.119501	0.157419
Worst	3.06422	3.2337	3.46239	3.11708
St. dev.	0.63374	1.02559	0.800209	0.792617
Median	0.510297	0.449116	0.542298	0.492164

Table 7.18: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for Osyczka2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.10 Osyczka2's Test Function

Figures C.28, C.29 and C.30 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in Osyczka2's test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.18 and 7.19 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. In this test function, all the algorithms seem to have a similar behavior.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0	0.0333333	0.433333	0.366667
Best	0	1	2	2
Worst	0	0	0	0
St. dev.	0	0.182574	0.626062	0.614948
Median	0	0	0	0

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.133333	0.266667	0.866667	0.566667
Best	2	3	7	3
Worst	0	0	0	0
St. dev.	0.434172	0.691492	1.4077	0.935261
Median	0	0	0	0

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.266667	0.433333	1.9	1.5
Best	3	2	5	7
Worst	0	0	0	0
St. dev.	0.691492	0.678911	1.56139	1.85231
Median	0	0	2	1

Table 7.19: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for Osyczka2's test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	1.84155	1.97117	2.04406	1.54913
Best	0.383413	0.0750232	0.0917061	0.641903
Worst	3.68453	6.23119	9.23048	2.45544
St. dev.	0.817318	1.2602	1.75107	0.489508
Median	1.71813	1.89949	1.69158	1.65093

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	1.2537	1.15029	0.937352	0.903529
Best	0.515482	0.491596	0.0620102	0.283415
Worst	2.13084	2.51154	1.90621	2.61298
St. dev.	0.347702	0.441866	0.369458	0.436752
Median	1.27737	1.13153	0.904365	0.811053

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0.559094	0.468305	0.558566	0.415296
Best	0.23213	0.197503	0.184375	0.240033
Worst	1.10088	1.22586	1.34691	1.32692
St. dev.	0.238506	0.238177	0.272548	0.195451
Median	0.481325	0.403037	0.471071	0.396541

Table 7.20: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Generational Distance metric for the Speed Reducer test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.11 Speed Reducer Test Function

Figures C.31, C.32 and C.33 show the graphical results produced by a) fixed adaptation, b) random adaptation, c) self adaptation and d) self-adaptation using half of the parameters' range in the Speed Reducer test function. The solutions displayed correspond to the merge of the final results obtained from all the executions performed.

Tables 7.20 and 7.21 show the comparison of results among the four algorithms considering the generational distance and the success counting metrics, respectively. In this test function, Fixed produced the best results with respect to the generational distance metric. However, the self-adaptive (half of the range) produced the best results for the Success counting metric.

50 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0	0	0	0
Best	0	0	0	0
Worst	0	0	0	0
St. dev.	0	0	0	0
Median	0	0	0	0

100 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0	0	0	0
Best	0	0	0	0
Worst	0	0	0	0
St. dev.	0	0	0	0
Median	0	0	0	0

250 iterations

Statistics	Approaches: Fixed	Random	Self-adaptive	Self-adaptive half of the range
Mean	0	0.0333333	0	0
Best	0	1	0	0
Worst	0	0	0	0
St. dev.	0	0.182574	0	0
Median	0	0	0	0

Table 7.21: Comparison of results of the approaches for parameter tuning (fixed, random, self-adaptive and self-adaptive using half of the parameters' range) with respect to the Success Counting metric for the Speed Reducer test function. In this comparison, 50, 100, and 250 iterations of the algorithm performed.

7.6.12 Conclusions from the Experiment

Although the results obtained from this experiment seem inconclusive, we argue that the use of a self-adaptation mechanism for adjusting the parameters gives a better performance in a wider range of functions. In this experiment, most of the time, the approach which performed best for the 25 generations, was also the best performer when adopting 50 and 250 generations. So, we argue, that the proposal to self-adapt the parameters improved the overall performance of the algorithm. Thus, we advise to use this sort of self-adaptation mechanism, if dealing with an unknown problem. However, if the best values for W, C1, and C2 are known in advance for a certain problem, then obviously they must be used instead. Since the self-adaptation that uses a half of the range methodology had the best performance in our study, we decided to adopt this methodology as part of our algorithm.

After introducing this last component (i.e., the self-adaptation mechanism), the final version of our proposed algorithm is shown in Algorithm 31. This approach does not require any manual fine-tuning of its PSO parameters.

Algorithm 31 MOPSO - final step

Require: $nsubswarms, nparticles, nconst$ and $dimensions$

$WSET \leftarrow \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
$C1SET \leftarrow C2SET \leftarrow \{1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0\}$
$WSUM \leftarrow C1SUM \leftarrow C2SUM \leftarrow \{1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0\}$
$\vec{gbest} \leftarrow \vec{x_0}$
$GBEST_{final} \leftarrow \emptyset$
for $s = 0$ to $nsubswarms$ **do**
  $GBEST^s \leftarrow \emptyset$
  for $i = 0$ to $nparticles/nsubswarms$ **do**
    $\vec{pbest}_i^s \leftarrow \vec{x_i^s} \leftarrow initialize_randomly()$
    $fitness_i^s \leftarrow f(\vec{x_i^s})$
    $velocity_{id}^s \leftarrow 0$
    if $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{gbest}))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{gbest})\&\&(\neg\exists\ \vec{y}^* \in GBEST^s \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness_i^s})))$ **then** $GBEST^s \leftarrow GBEST^s \cup \vec{x_i^s}$
  end for
end for
repeat
  $GBEST_{temp} \leftarrow \emptyset$
  for $s = 0$ to $nsubswarms$ **do**
    for $i = 0$ to $nparticles$ **do**
      $indC1 \leftarrow indC2 \leftarrow indW \leftarrow 0$
      $randtemp \leftarrow U(0, \sum_0^{|C1SUM|} C1SUM)$
      while $randtemp > 0$ **do** $randtemp \leftarrow randtemp - C1SUM_{indC1++}$
      $randtemp \leftarrow U(0, \sum_0^{|C2SUM|} C2SUM)$
      while $randtemp > 0$ **do** $randtemp \leftarrow randtemp - C2SUM_{indC2++}$
      $randtemp \leftarrow U(0, \sum_0^{|WSUM|} WSUM)$
      while $randtemp > 0$ **do** $randtemp \leftarrow randtemp - WSUM_{indW++}$
      if flip(0.5)**then** $gbest \leftarrow GBEST^s_{U(0,|GBEST^s|)}$
      else $gbest \leftarrow y | y \in GBEST^s \&\& \nexists\ z \in GBEST^s | z - x_i^s < y - x_i^s$
      if x^s is equal $gbest$ **then** turbulence (x^s)
      for $d = 0$ to $ndimensions$ **do**
        $velocity_{id}^s \leftarrow WSET_{indW} \times velocity_{id}^s + C1SET_{indC1} \times U(0,1) \times (pbest_{id}^s - x_{id}^s) + C2SET_{indC2} \times U(0,1) \times (gbest - x_{id}^s)$
        $x_{id}^s \leftarrow x_{id}^s + velocity_{id}^s$
      end for
      $fitness_i^s \leftarrow f(\vec{x_i^s})$
      if $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{pbest}_i^s))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{pbest}_i^s)\&\&fitness_i^s$ is $\sim$ to $f(\vec{pbest}_i^s))$ **then** $\vec{pbest}_i^s \leftarrow \vec{x_i^s}$
      if $(\sum_0^{nconst} violation(\vec{x_i}) < \sum_0^{nconst} violation(\vec{gbest}))||(\sum_0^{nconst} violation(\vec{x_i}) == \sum_0^{nconst} violation(\vec{gbest})\&\&(\neg\exists\ \vec{y}^* \in GBEST^s \mid \vec{f}(\vec{y}^*) \preceq \vec{f}(\vec{fitness_i^s})))$ **then** $GBEST^s \leftarrow GBEST^s \cup \vec{x_i^s}$
    end for
    $GBEST_{final} \leftarrow GBEST_{final} \cup GBEST^s$
  end for
  $GBEST_{final} \leftarrow approach - to - distribute - solutions(GBEST_{final})$
  group $GBEST_{final}$ into $nsubswarms$
  for $s = 0$ to $nsubswarms$ **do** $GBEST_s \leftarrow$ randomly select a group formed in last step
until Termination criterion

TSC	MOPSO	Mostaghim's MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.340986	0.454932	0.29932
Mostaghim's MOPSO	0.240833	———-	0.42	0.196667
NSGA-II	0.479167	0.5225	———-	0.513333
ϵ-MOEA	0.575125	0.556761	0.671536	———-

Table 7.22: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Kursawe's test function.

7.7 Comparison of Results

In this section, we compare the results of the final version of our algorithm against the NSGA-II, ϵ-MOEA and against another multiobjective particle swarm optimizer (Mostaghim's MOPSO [91]).

In the following examples, the NSGA-II was run using a population size of 40, a crossover rate of 0.8 (uniform crossover was adopted), tournament selection, and a mutation rate of $1/N$, where N = number of variables (real representation was adopted), a distribution index of 15 for real-coded crossover, a mutation rate of $1/L$, and a mutation rate of $1/N$, where N = number of variables, a distribution index of 20 for real-coded mutation. The ϵ-MOEA was run using a population size of 40, a crossover rate of 0.8 (uniform crossover was adopted), distribution index of 15 for real-coded crossover and distribution index of 20 for real-coded mutation. Mostaghim's MOPSO was run using a maximum number of cycles of 50, a population size of 40, an archive size of 40, a number of parameters of 30 and a turbulence factor of 0.01. Our MOPSO used 40 particles and a total of 8 swarms. Two versions of our MOPSO were used in this comparison, one of which uses self-adaptation.

The total number of fitness function evaluations was set to $2,000$ for all the algorithms compared (50 generations). Since Mostaghim's approach doesn't have a constraint-handling technique (and in order to avoid any bias in the comparison of results), we only compared results with respect to this approach in unconstrained test functions.

7.7.1 Kursawe's Test Function

Figure 7.2 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) our MOPSO in the first test function chosen. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.22, 7.23 and 7.24, show the comparison of results among the three algorithms considering two set coverage, hyper volume, inverted generational distance and the success counting metrics. In this test function, ϵ-MOEA performed better than the others. However, it is important to note that our MOPSO was the only algorithm which could discover the (-20,0) point. This is important, because it is quite difficult for most MOEAs to generate isolated point is quite difficult to be reach by any multi-objective evolutionary algorithm (in fact, neither the NSGA-II nor ϵ-MOEA could generate this point), which is a good indicative of the capabilities of our approach to reach disconnected regions of a Pareto front.

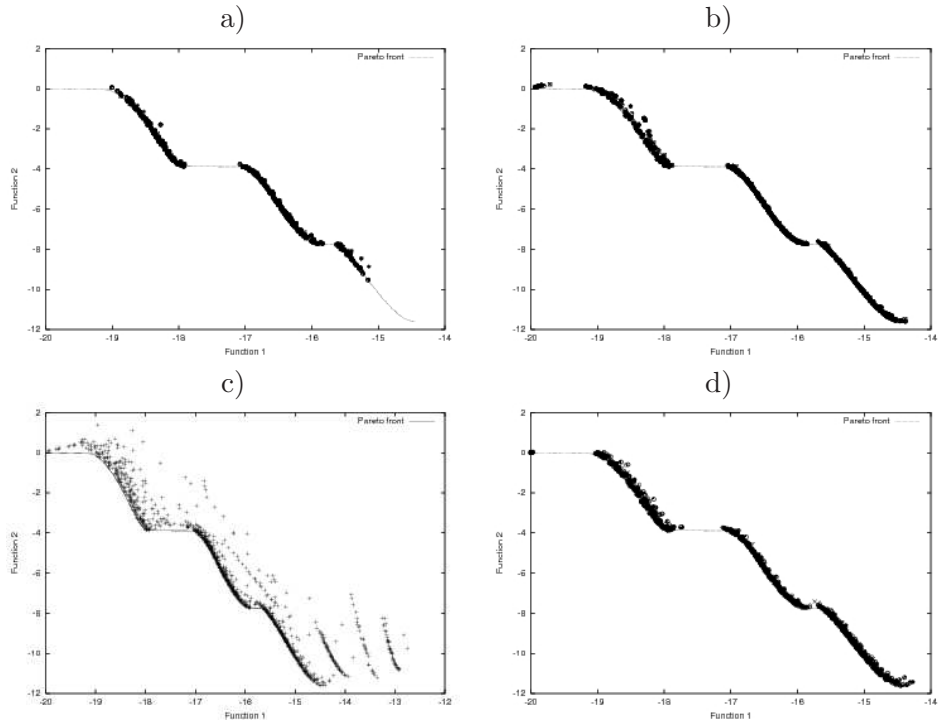

Figure 7.2: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for Kursawe's test function.

HV	**MOPSO**	**Mostaghim's MOPSO**	**NSGA-II**	**ϵ-MOEA**
MOPSO	———-	0.188373	1.01789	0.170263
Mostaghim's MOPSO	0.0571032	———-	0.894887	0.0717263
NSGA-II	0.0537395	0.111614	———-	0.0796734
ϵ-MOEA	0.0774617	0.111677	1.11986	———-

Table 7.23: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for Kursawe's test function.

		Approaches			
	Statistics	**MOPSO**	**Mostaghim's MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	**Mean**	0.00801815	0.02533	0.0825022	0.00602655
	Best	0.00681556	0.00757799	0.0588314	0.00405783
	Worst	0.011734	0.0753964	0.117137	0.00973392
	St. dev.	0.0011151	0.02107	0.021286	0.00161515
	Median	0.00753431	0.0182831	0.078836	0.00546673
SC	**Mean**	9.56667	8.8	15.5	40.5667
	Best	18	19	22	49
	Worst	4	0	8	33
	St. dev.	underline3.74795	6.5147	4.84056	5.69139
	Median	4.5	16	20	43

Table 7.24: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for Kursawe's test function.

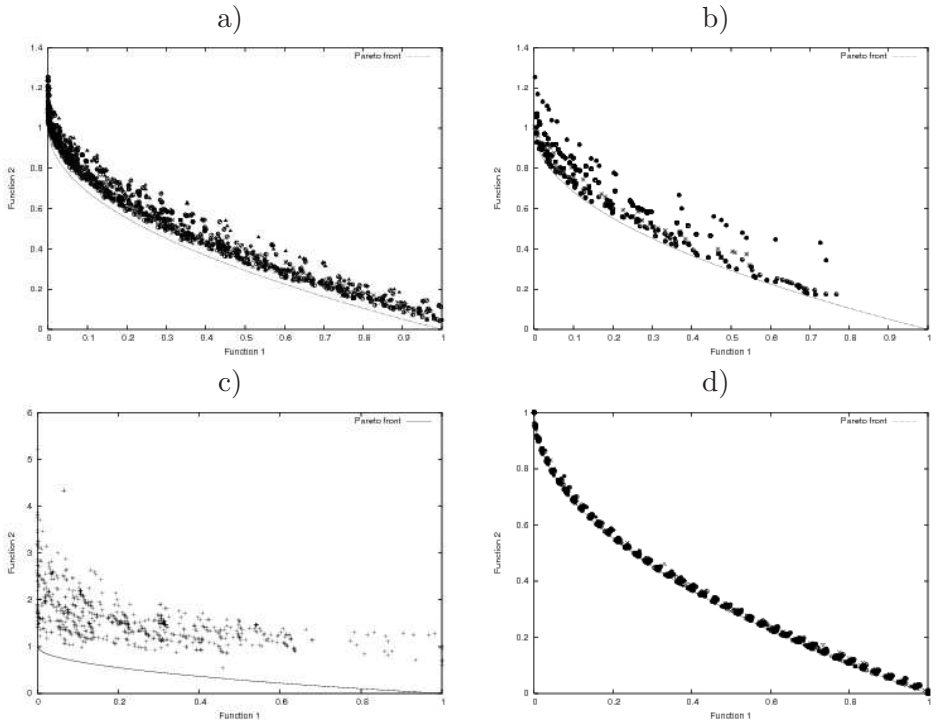

Figure 7.3: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for ZDT1's test function.

TSC	MOPSO	Mostaghim's MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———	1	1	0.983333
Mostaghim's MOPSO	0	———	0	0.0607407
NSGA-II	0	1	———	0.169062
ϵ-MOEA	0.00605144	0.993949	0.202723	———

Table 7.25: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT1's test function.

7.7.2 ZDT1's Test Function

Figure 7.3 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA, Mostaghim's MOPSO and d) our MOPSO in ZDT1's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.25, 7.26 and 7.27, show the comparison of results among the three algorithms considering the following metrics: two set coverage, hyper volume, inverted generational distance and the success counting. In this test function, our MOPSO was the algorithm which performed best with respect to all metrics.

HV	MOPSO	Mostaghim's MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.406051	0.0402556	0.0461719
Mostaghim's MOPSO	0	———-	0	2.37284E-06
NSGA-II	0	0.362969	———-	0.0164115
ϵ-MOEA	1.04759e-05	0.253873	0.0110925	———-

Table 7.26: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT1's test function.

		Approaches			
	Statistics	**MOPSO**	**Mostaghim's MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	**Mean**	0.000984289	0.0601869	0.00676222	0.0227843
	Best	0.000826707	0.028702	0.00261635	0.00640436
	Worst	0.00118531	0.0877538	0.0169496	0.0379035
	St. dev.	9.26088E-05	0.0125145	0.0045472	0.00851036
	Median	0.000978502	0.0623778	0.0049536	0.0240974
SC	**Mean**	9.9	0	0.0666667	0.0666667
	Best	20	0	2	1
	Worst	6	0	0	0
	St. dev.	3.4074	0	0.365148	0.253708
	Median	6	0	0	0

Table 7.27: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for ZDT1's test function.

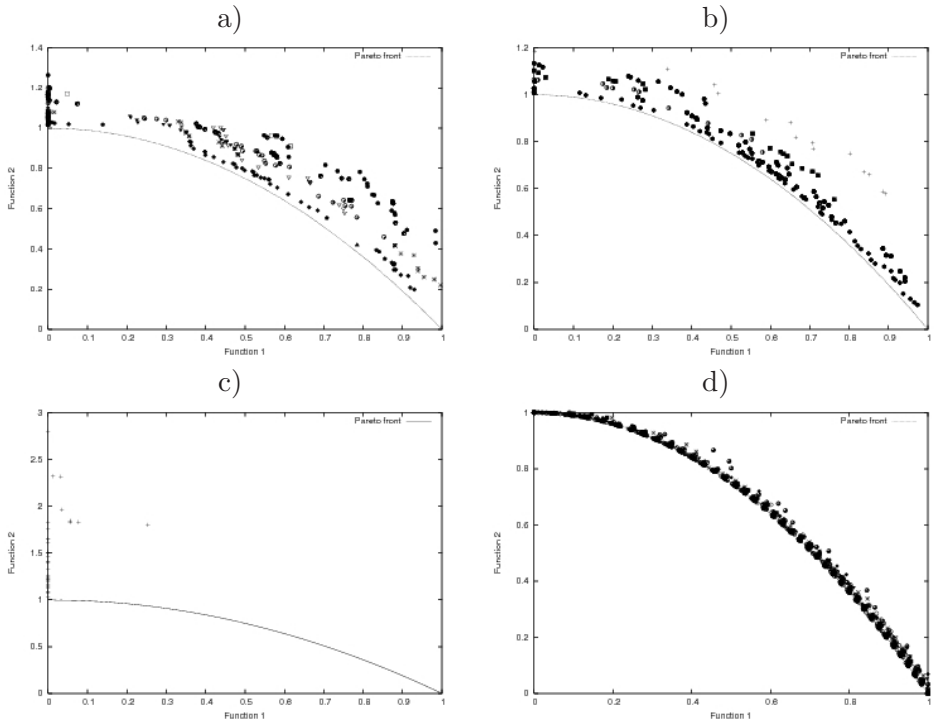

Figure 7.4: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for ZDT2's test function.

TSC	MOPSO	Mostaghim's MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———	1	0.995726	0.987179
Mostaghim's MOPSO	0	———	0	0
NSGA-II	0	0.788618	———	0.0704607
ϵ-MOEA	0	0.864826	0.603198	———

Table 7.28: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for ZDT2's test function.

7.7.3 ZDT2's Test Function

Figure 7.4 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) our MOPSO in ZDT2's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.28, 7.29 and 7.30, show the comparison of results among the three algorithms considering the following metrics: two set coverage, hyper volume, inverted generational distance and the success counting. In this test function, our MOPSO was the algorithm which performed best with respect to all metrics.

HV	MOPSO	Mostaghim's MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.355323	0.0636189	0.034894
Mostaghim's MOPSO	0	———-	0	0
NSGA-II	0	0.23358	———-	0.00145575
ϵ-MOEA	0	0.296025	0.0277186	———-

Table 7.29: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for ZDT2's test function.

		Approaches			
	Statistics	**MOPSO**	**Mostaghim's MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	**Mean**	0.00261164	0.0672163	0.0322927	0.0166433
	Best	0.000783843	0.0520029	0.00560298	0.00200895
	Worst	0.0329462	0.0931413	0.0573697	0.0514464
	St. dev.	0.00650816	0.0117475	0.0188163	0.0143003
	Median	0.000911632	0.0634132	0.0294469	0.0130747
SC	**Mean**	23.7	0	0	0
	Best	41	0	0	0
	Worst	4	0	0	0
	St. dev.	13.9189	0	0	0
	Median	37	0	0	0

Table 7.30: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for ZDT2's test function.

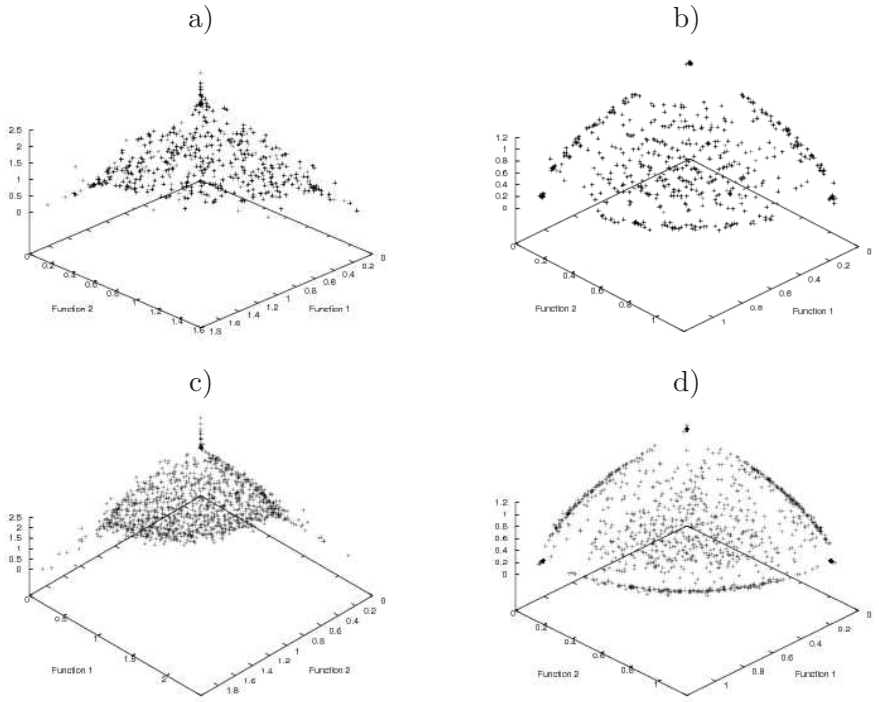

Figure 7.5: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) MOPSO, for DTLZ2's test function.

TSC	MOPSO	Mostaghim's MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.938849	0.903777	0.947855
Mostaghim's MOPSO	0.249167	———-	0.574167	0.653333
NSGA-II	0.42471	0.897297	———-	0.915058
ϵ-MOEA	0.0708333	0.436667	0.418333	———-

Table 7.31: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for DTLZ2's test function.

7.7.4 DTLZ2's Test Function

Figure 7.5 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA, c) Mostaghim's MOPSO and d) our MOPSO in DTLZ2's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.31, 7.32 and 7.33, show the comparison of results among the three algorithms considering the following metrics: two set coverage, hyper volume, inverted generational distance and the success counting. In this test function, our MOPSO was the algorithm which performed best with respect to all metrics.

HV	MOPSO	Mostaghim's MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.057652	0.057310	0.065215
Mostaghim's MOPSO	0	———-	0.020639	0.038299
NSGA-II	0.001073	0.021591	———-	0.040435
ϵ-MOEA	0.002234	0.032507	0.033691	———-

Table 7.32: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for DTLZ2's test function.

		Approaches			
	Statistics	**MOPSO**	**Mostaghim's MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	Mean	0.00063264	0.000663823	0.000694023	0.000640918
	Best	0.000550767	0.000569451	0.000613578	0.000550314
	Worst	0.000743347	0.000855971	0.000821955	0.000750676
	St. dev.	4.48938E-05	6.12624E-05	5.01947E-05	4.69864E-05
	Median	0.000631137	0.000649068	0.00071043	0.000636023
SC	**Mean**	6.46667	4.6	3.9	3.36667
	Best	12	11	9	7
	Worst	2	1	1	1
	St. dev.	2.38867	2.22215	2.4119	2.17324
	Median	6	4	3	2

Table 7.33: Comparison of results of our MOPSO, Mostaghim's MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for DTLZ2's test function.

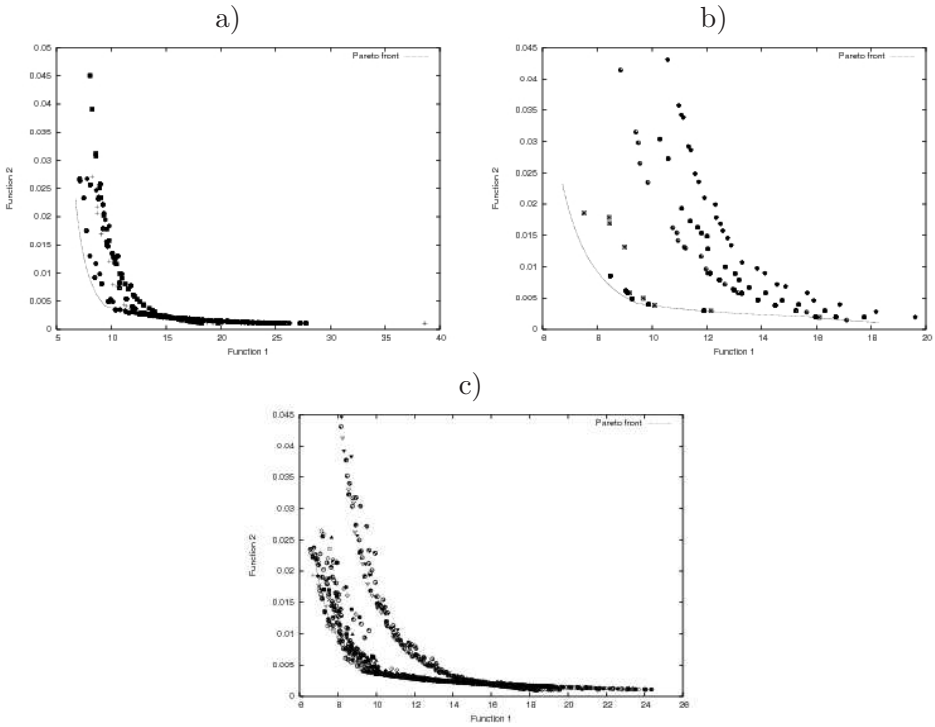

Figure 7.6: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for the Welded Beam test function.

TSC	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.438863	0.699526
NSGA-II	0.0433333	———-	0.5925
ϵ-MOEA	0	0.139241	———-

Table 7.34: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for the Welded Beam test function.

7.7.5 Welded Beam Test Function

Figure 7.6 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in the Welded Beam test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.34, 7.35 and 7.36, show the comparison of results among the three algorithms considering the following metrics: two set coverage, hyper volume, inverted generational distance and the success counting. In this test function, our MOPSO was the algorithm which performed best with respect to all metrics.

HV	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.0217217	0.0210309
NSGA-II	2.19062e-05	———-	0.0063767
ϵ-MOEA	0	0.00391661	———-

Table 7.35: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for the Welded Beam test function.

		Approaches		
	Statistics	**MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	**Mean**	0.00710279	0.0106717	0.0365222
	Best	0.00243346	0.00469789	0.017034
	Worst	0.0188052	0.0196519	0.0625128
	St. dev.	0.00454798	0.00497636	0.0167276
	Median	0.00482416	0.00989811	0.0358428
SC	**Mean**	1.76667	0	0
	Best	1	0	0
	Worst	3	0	0
	St. dev.	0.678911	0	0
	Median	2	0	0

Table 7.36: Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for the Welded Beam test function.

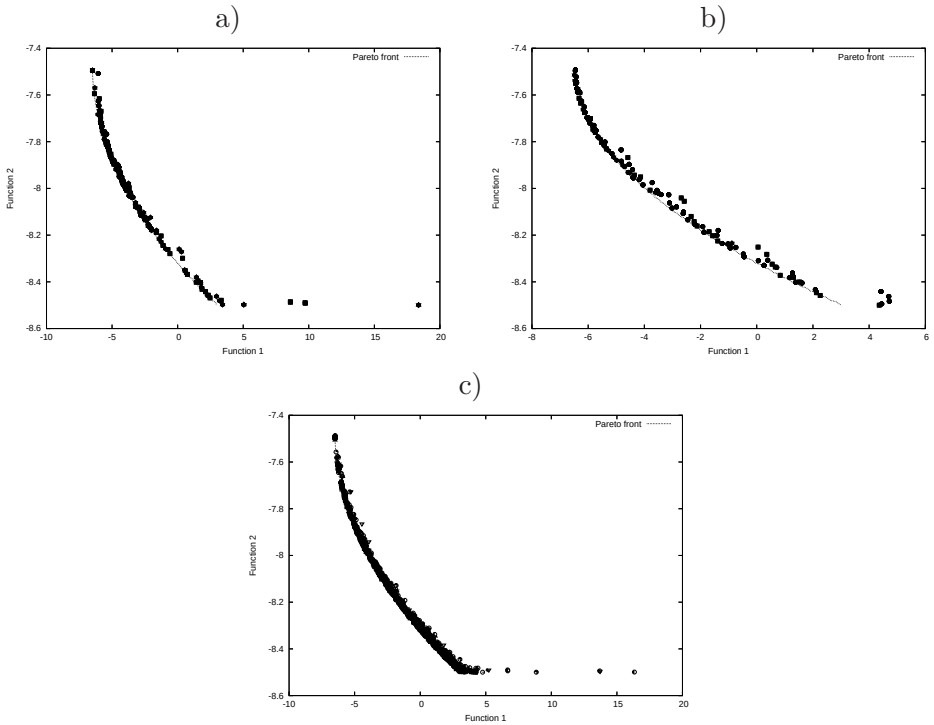

Figure 7.7: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Kita's test function.

TSC	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.592593	0.674264
NSGA-II	0.275833	———-	0.6025
ϵ-MOEA	0.320423	0.545775	———-

Table 7.37: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Kita's test function.

7.7.6 Kita's Test Function

Figure 7.7 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in Kita's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.38, 7.37 and 7.39, show the comparison of results among the three algorithms considering the following metrics: two set coverage, hyper volume, inverted generational distance and the success counting. In this test function, our MOPSO was the algorithm which performed best with respect to all metrics.

HV	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.171587	0.192213
NSGA-II	0.00474455	———-	0.115128
ϵ-MOEA	0.00739919	0.0940972	———-

Table 7.38: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for Kita's test function.

		Approaches		
	Statistics	**MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	**Mean**	0.00785754	0.0828163	0.03697
	Best	0.00593717	0.00936987	0.0134287
	Worst	0.0132567	0.197385	0.0833982
	St. dev.	0.00161361	0.0721825	0.0310189
	Median	0.00734665	0.119276	0.0203783
SC	**Mean**	8.26667	11	7.2
	Best	15	15	14
	Worst	3	7	4
	St. dev.	2.89986	3.2056	3.63318
	Median	5	11	5

Table 7.39: Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for Kita's test function.

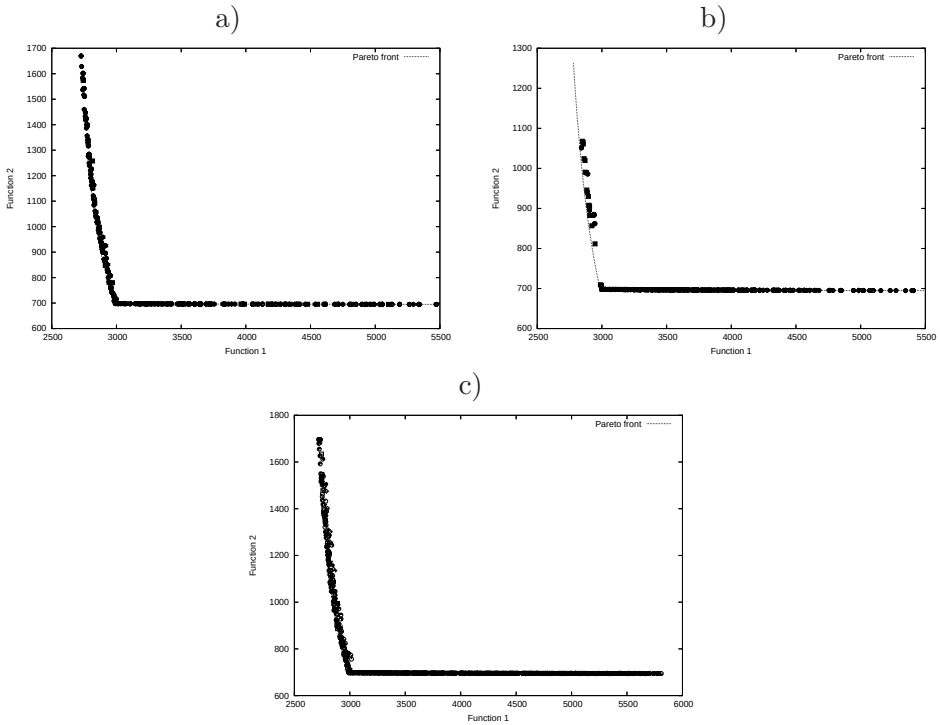

Figure 7.8: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Speed reducer's test function.

TSC	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	0.362543	0.481959
NSGA-II	0.354167	———-	0.670833
ϵ-MOEA	0.409982	0.465241	———-

Table 7.40: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for the Speed Reducer test function.

7.7.7 Speed Reducer Test Function

Figure 7.8 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in the Speed Reducer test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.40, 7.41 and 7.42, show the comparison of results among the three algorithms considering the following metrics: two set coverage, hyper volume, inverted generational distance and the success counting metrics. In this test function, our MOPSO produced the best results with respect to the two set coverage

HV	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———-	3646.23	52758
NSGA-II	933.867	———-	46972.6
ϵ-MOEA	34.1542	41.9211	———-

Table 7.41: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for the Speed Reducer test function.

		Approaches		
	Statistics	**MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	**Mean**	1.77744	9.41548	6.65309
	Best	0.56863	2.01383	5.5782
	Worst	19.2329	17.222	10.3793
	St. dev.	3.80189	6.49236	1.59788
	Median	{underline0.703888	9.61791	5.62359
SC	**Mean**	0	0	0
	Best	0	0	0
	Worst	0	0	0
	Stdev	0	0	0
	Median	0	0	0

Table 7.42: Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for the Speed Reducer test function.

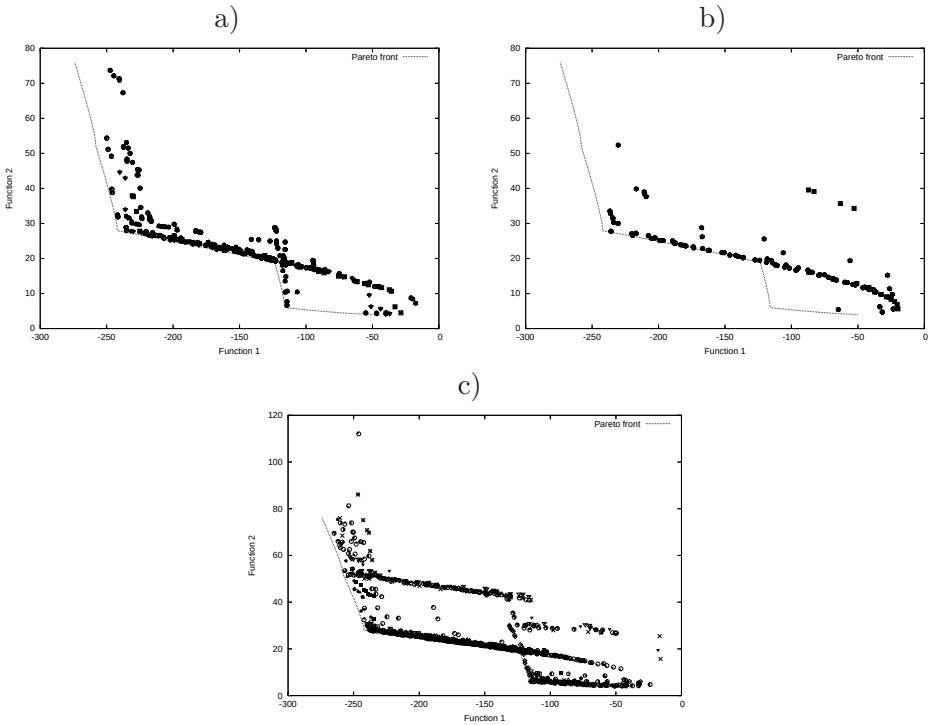

Figure 7.9: Pareto fronts produced by a) NSGA-II, b) ϵ-MOEA, and c) MOPSO, for Osyczka2's test function.

TSC	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———	0.454274	0.548708
NSGA-II	0.0175	———	0.26
ϵ-MOEA	0.0498339	0.548708	———

Table 7.43: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Two Set Coverage metric for Osyczka2's test function.

7.7.8 Osyczka2's Test Function

Figure 7.9 shows the graphical results produced by a) the NSGA-II, b) ϵ-MOEA and c) our MOPSO in Osyczka2's test function. The true Pareto front of the problem is shown as a continuous line. The solutions displayed correspond to the merge of the final results obtained from all the executions performed. Tables 7.43, 7.44 and 7.45, show the comparison of results among the three algorithms considering the following metrics: two set coverage, hyper volume, inverted generational distance and the success counting. In this test function, our MOPSO was the algorithm which performed best with respect to all metrics.

7.8 Conclusions

Based on the analysis of the experiments and comparisons performed in this chapter, we conclude that the use a self-adaptive scheme to modify the parameter's values of our

HV	MOPSO	NSGA-II	ϵ-MOEA
MOPSO	———	370.676	1269.45
NSGA-II	10.945	———-	733.396
ϵ-MOEA	0.270441	26.6716	———-

Table 7.44: Comparison of results of our MOPSO, NSGA-II, and ϵ-MOEA with respect to the Hyper Volume metric for Osyczka2's test function.

		Approaches		
	Statistics	**MOPSO**	**NSGA-II**	**ϵ-MOEA**
IGD	**Mean**	0.973828	1.60855	2.02818
	Best	0.277941	0.575352	1.19288
	Worst	4.72277	5.03231	6.40672
	St. dev.	0.782608	1.42304	1.76107
	Median	0.760223	1.12498	1.27526
SC	**Mean**	0.366667	0	0
	Best	2	0	0
	Worst	0	0	0
	St. dev.	0.614948	0	0
	Median	0	0	0

Table 7.45: Comparison of results of our MOPSO, NSGA-II and ϵ-MOEA with respect to the inverted generational distance (IGD) and Success counting (SC) for Osyczka2's test function.

MOPSO turns out to be beneficial in most cases, and only marginally harmful when the problem is very simple to solve. Note however, that since the difference is almost negligible, we recommend to use self-adaptation with our MOPSO in all cases.

8

Final Remarks

There were the main steps that we followed to reach the final version of our algorithm:

- We proposed an extension of the particle swarm optimization algorithm to handle multiobjective problems. The proposal uses sub-swarms, Pareto ranking and clustering techniques. This first version of the algorithm only handled unconstrained test problems, and was properly validated (using standard test functions and metrics).
- We developed a constraint-handling technique that was initially validated in a single objective version of the particle swarm optimization algorithm. Then, this constraint-handling technique was incorporated into our multiobjective particle swarm optimizer and the new version of the algorithm was validated again.
- In our final step, we performed a comprehensive study of the three most critical parameters of our multiobjective particle swarm optimization algorithm. Our study suggested that the design of a self-adaptation mechanism was the best choice in order to automatically adjust the parameters of our approach. This was the final version of our proposed algorithm, which was again validated using standard test functions and metrics.

8.1 Conclusions

Our main conclusions are the following:

- The particle swarm optimization algorithm is a viable alternative to solve multiobjective optimization problems.
- The selection of leaders was found to play a very significant role in the performance of our multiobjective particle swarm optimizer. This led us to experiment with different strategies for leader selection.

- We found that the use of subswarms promotes local search as an emergent behavior in our multiobjective particle swarm optimizer. Consequently, the performance of our approach was improved by the use of subswarms, particularly in the presence of disconnected Pareto fronts.
- The use of a relatively simple constraint-handling technique was sufficient to solve even problems in which the global optimum lies on the boundary between the feasible and the infeasible region.
- In general, we found that it is quite difficult to find fixed values for the three most significant parameters of our approach (W,C1 and C2). It is worth indicating that the comprehensive study of parameters done as part of this thesis is the first of its type (in the context of multiobjective particle swarm optimization). So, we designed a self-adaptation mechanism for these parameters, and we found this to be a good alternative to facilitate the use of our approach.
- The use of a perturbation mechanism in our multiobjective particle swarm optimizer was found to be critical to control its high selection pressure, as to avoid premature convergence.
- It is worth emphasizing that the final version of our proposed approach produced very competitive results while performing only $2,000$ fitness function evaluations. This is the lowest number of evaluations reported by any other multiobjective particle swarm optimizer to date.

8.2 Future Work

Some possible paths to extend this work are the following:

- Experiment with other PSO's models and with different interconnection topologies.
- Study alternative methods for the survivor selection mechanism.
- Study alternative (perhaps more elaborate) constraint-handling mechanisms.
- Study alternative mechanisms to accelerate convergence while keeping the same quality of results achieved in this thesis. Such an approach would be very useful to deal with real-world applications.

A

Four strategies to select one of the multiple gbest to apply in the PSO's velocity formula were proposed in Chapter 5, Section 4. This Appendix summarizes the results of such proposals in seven test functions [131]. Each algorithm were executed 30 times. The results from each strategy in each test function are graphically presented. Furthermore, six behavior statistics were taken at each generation of each algorithm's execution. Such statistics are: a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles. These statistics are meat to analyze the behavior of each gbest selection strategy.

A.1 ZDT1's test function

This test function is described in Section 2.5.5.

Figure A.1 shows the graphical results produced by each gbest selection strategy in ZDT1's test function. The true Pareto front of the problem is shown as a continuous line. Figure A.2 shows the statistics previously described for this test function.

A.2 ZDT2's test function

This test function is described in Section 2.5.6.

Figure A.3 shows the graphical results produced by each gbest selection strategy in ZDT2's test function. The true Pareto front of the problem is shown as a continuous line. Figure A.4 shows the statistics previously described for this test function.

A.3 ZDT3's test function

This test function is described in Section 2.5.7.

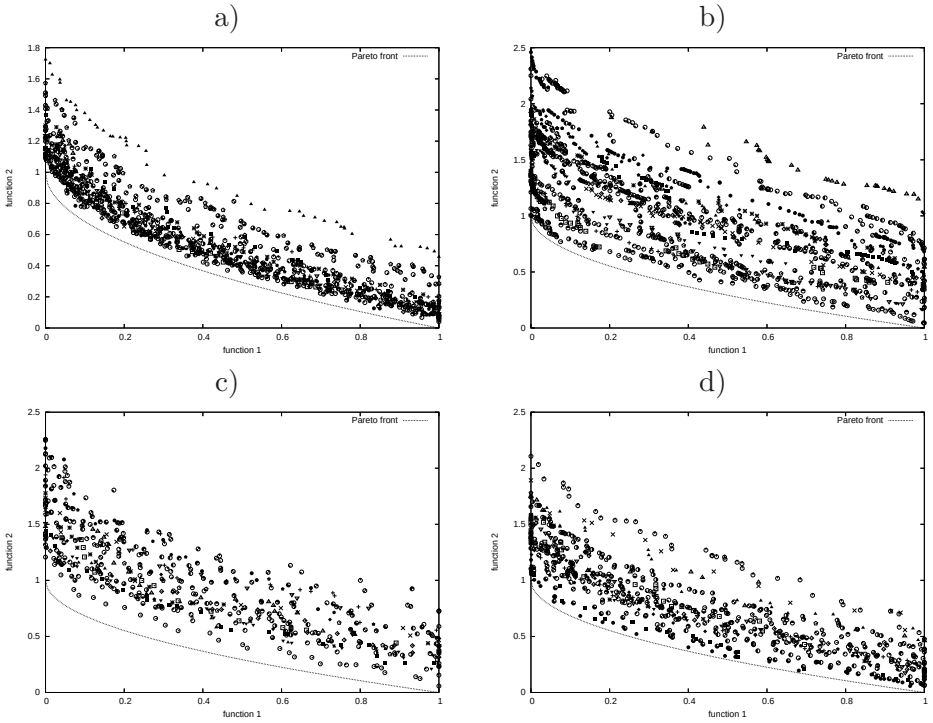

Figure A.1: Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZDT1's test function.

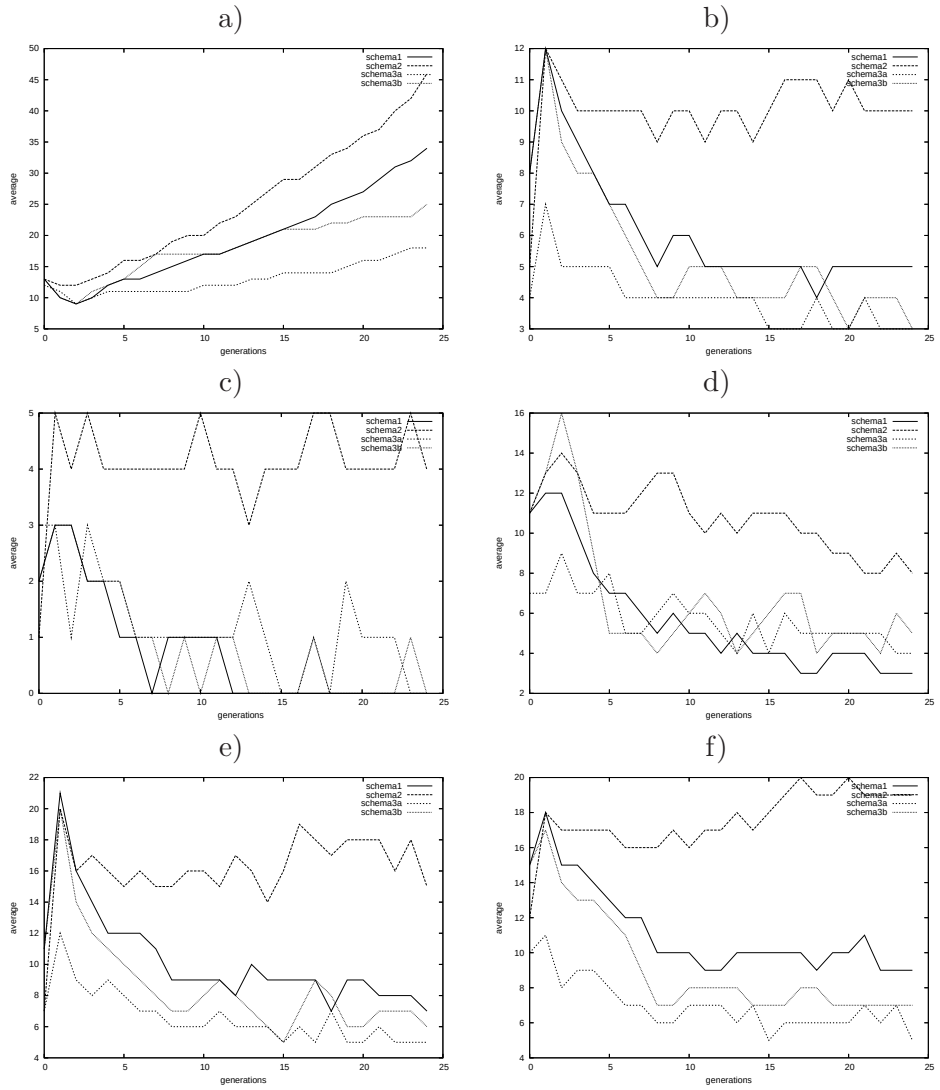

Figure A.2: Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles for ZDT1's test function.

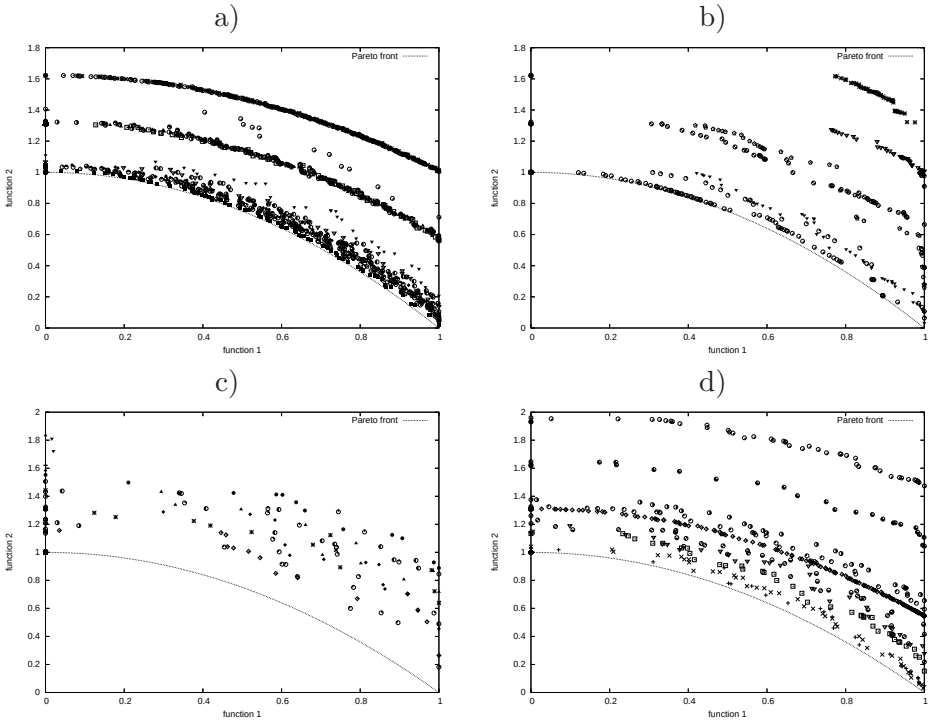

Figure A.3: Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZTD2's test function.

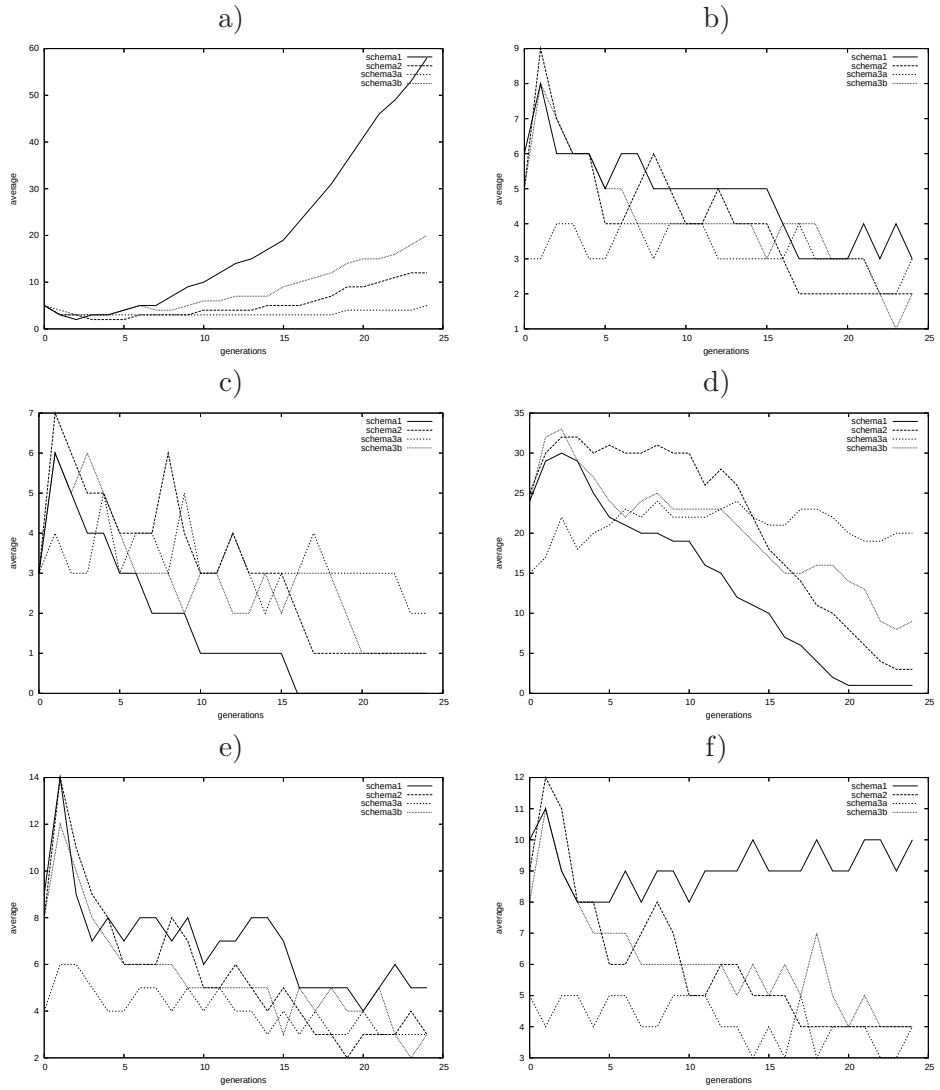

Figure A.4: Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particles' position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for ZDT2's test function.

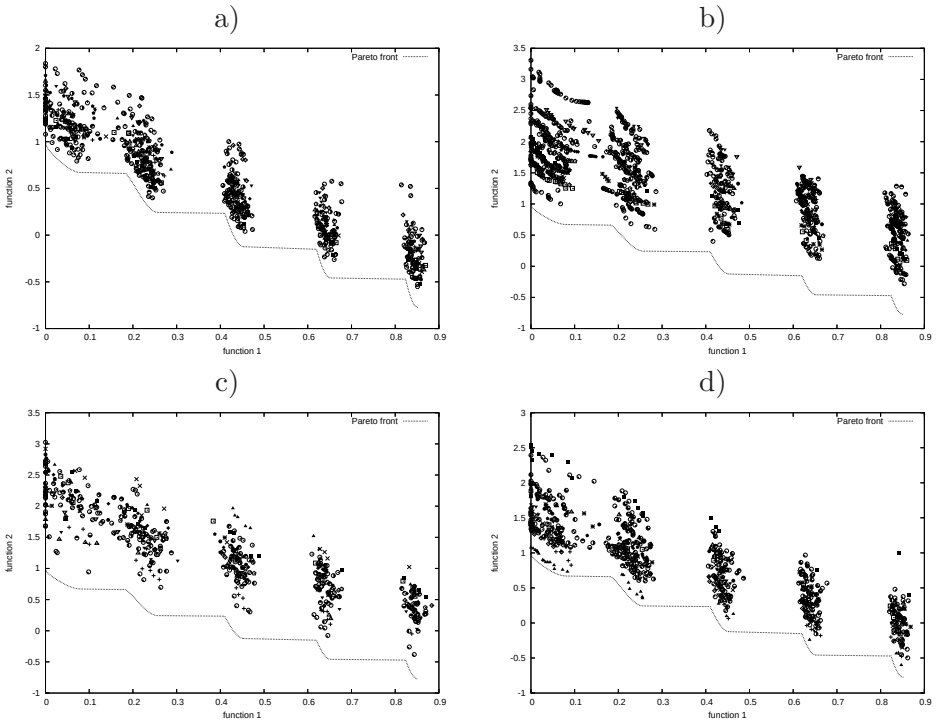

Figure A.5: Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZTD3's test function.

Figure A.5 shows the graphical results produced by each gbest selection strategy in ZDT3's test function. The true Pareto front of the problem is shown as a continuous line. Figure A.6 shows the statistics previously described for this test function.

A.4 ZDT6's test function

This test function is described in Section 2.5.9.

Figure A.7 shows the graphical results produced by each gbest selection strategy in ZDT6's test function. The true Pareto front of the problem is shown as a continuous line. Figure A.8 shows the statistics previously described for this test function.

A.5 Kursawe's test function

This test function is described in Section 2.5.3.

Figure A.9 shows the graphical results produced by each gbest selection strategy in Kursawe's test function. The true Pareto front of the problem is shown as a continuous line. Figure A.10 shows the statistics previously described for this test function.

A.6 Deb's test function

This test function is described in Section 2.5.1.

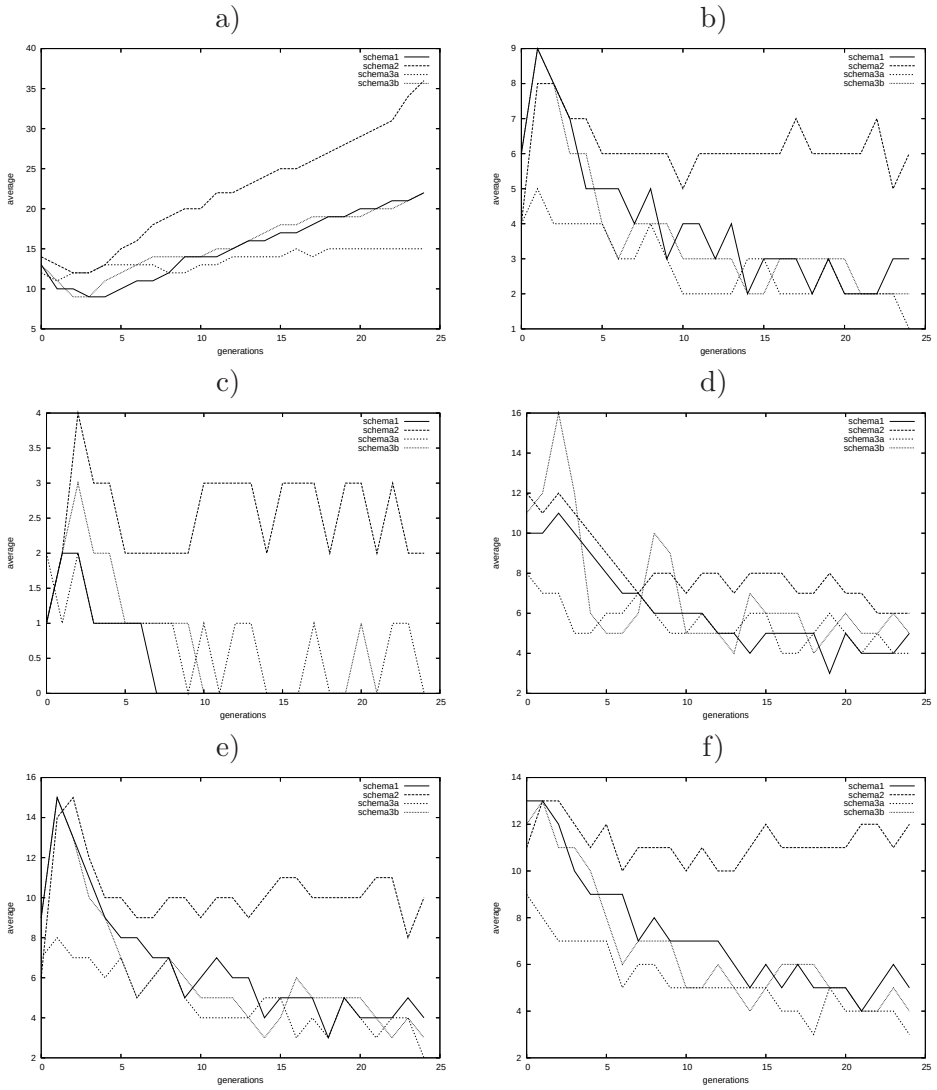

Figure A.6: Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for ZDT3's test function.

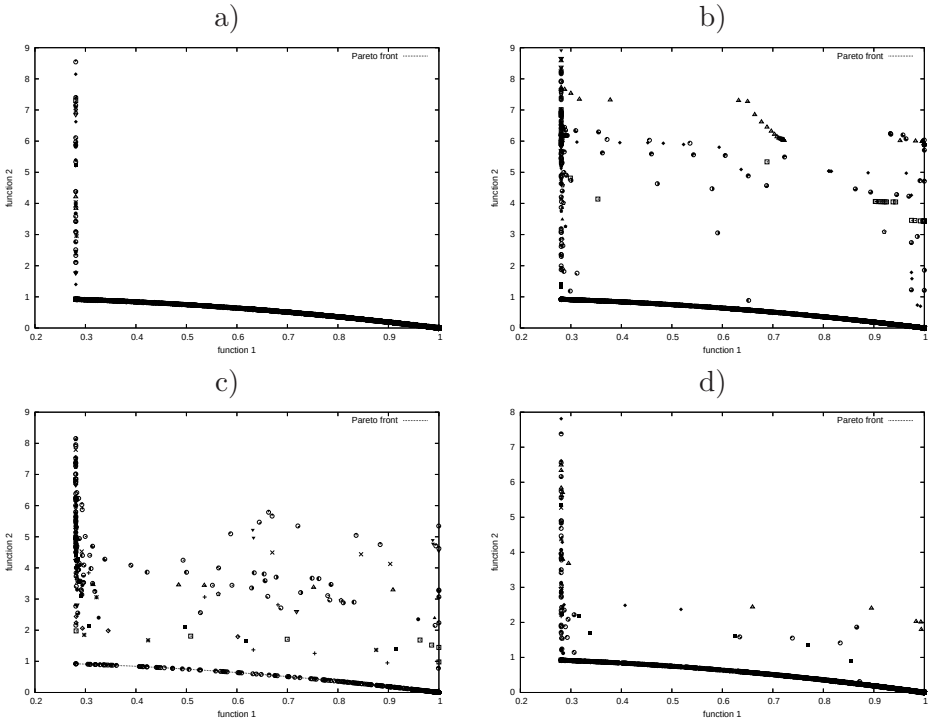

Figure A.7: Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for ZTD6's test function.

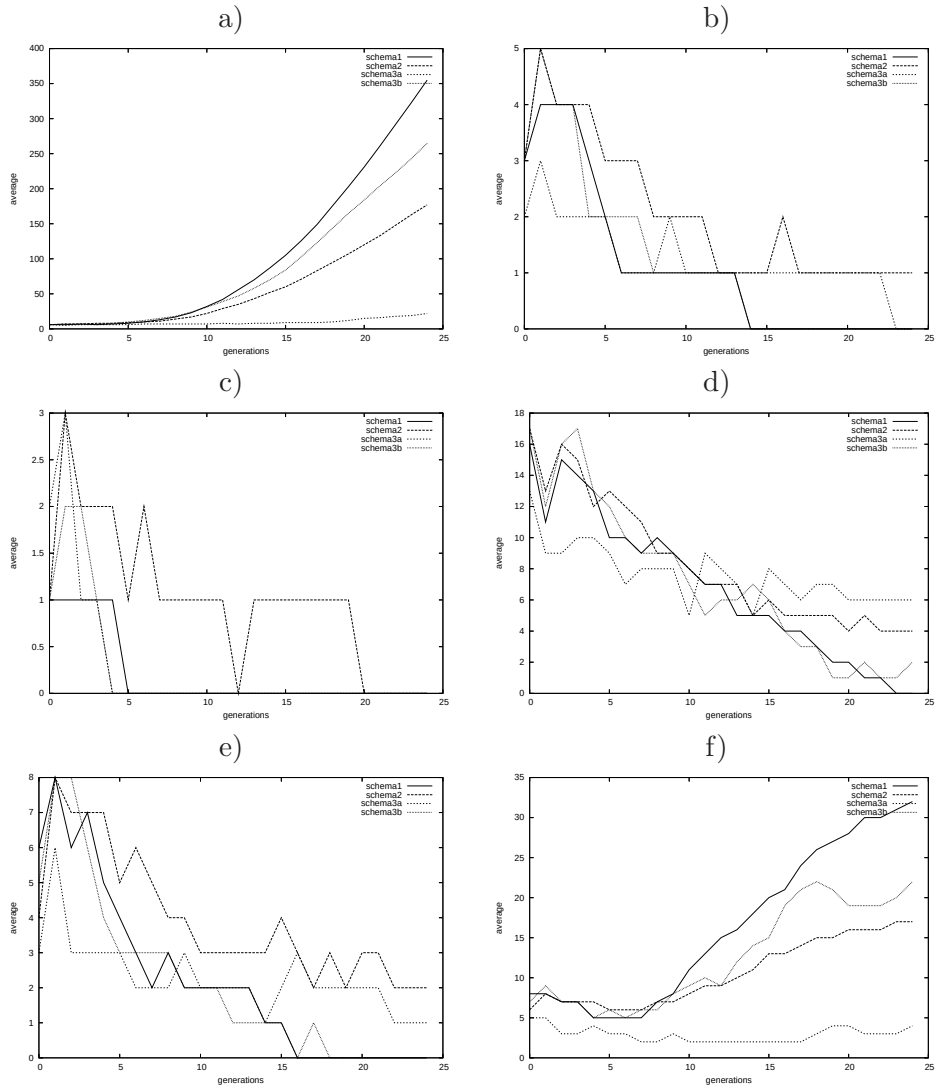

Figure A.8: Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for ZDT6's test function.

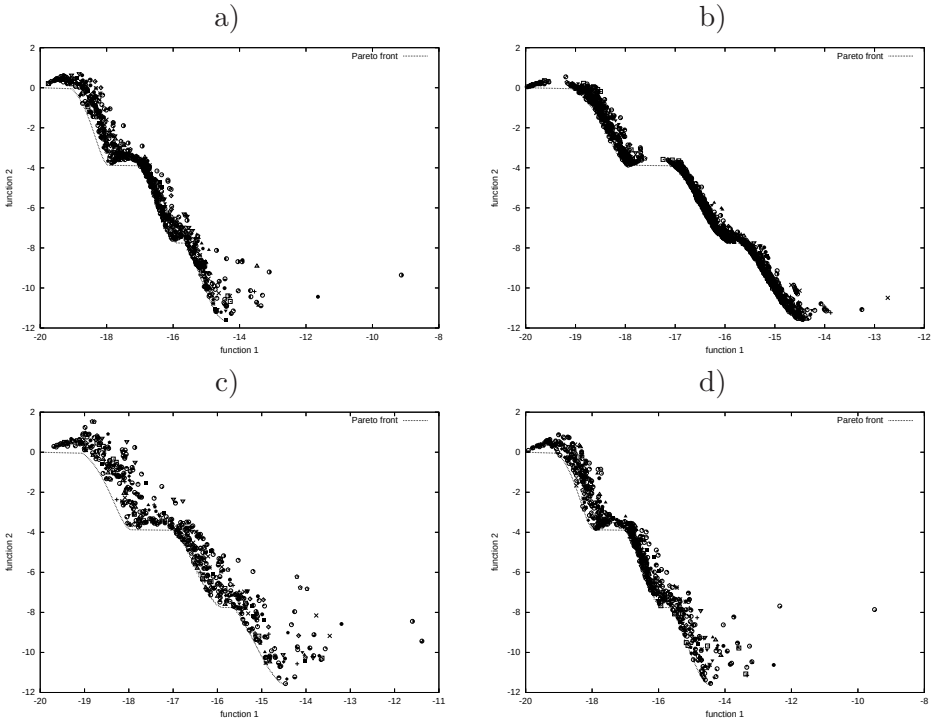

Figure A.9: Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for Kursawe's test function.

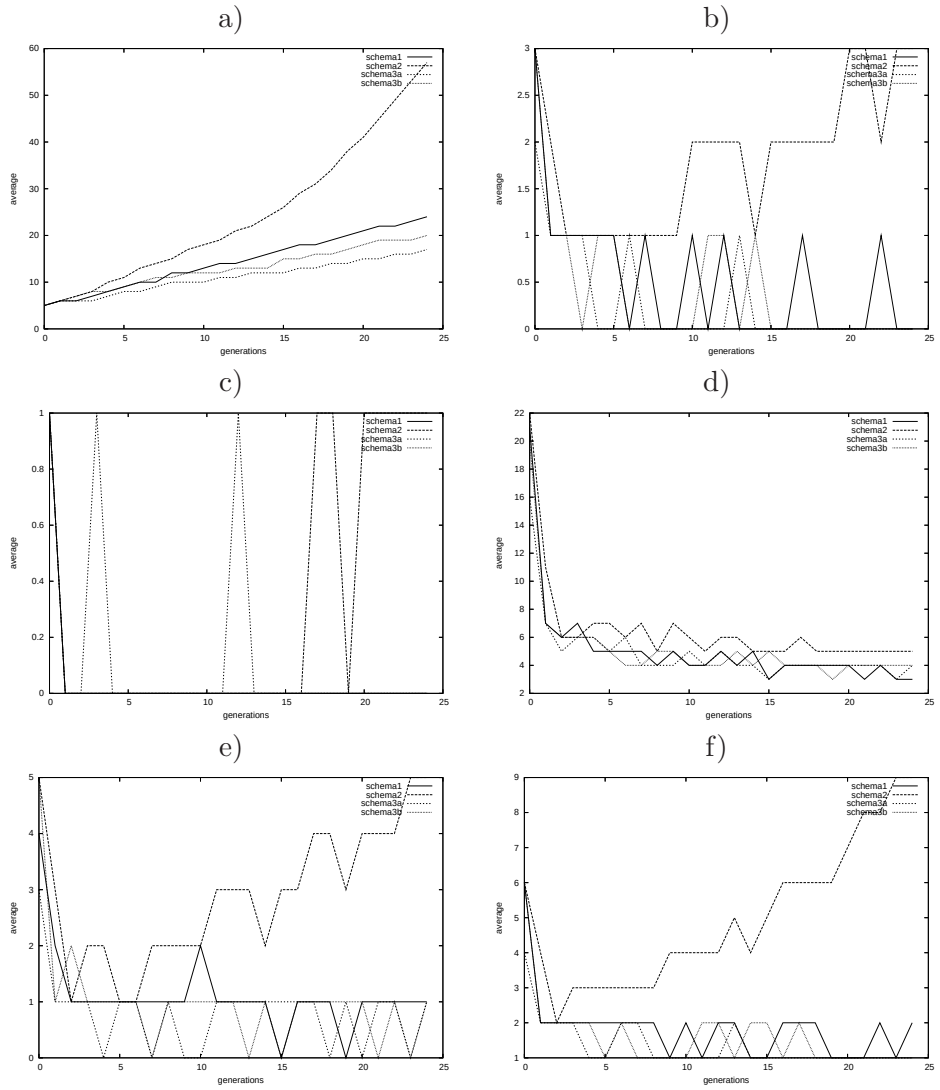

Figure A.10: Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for Kursawe's test function.

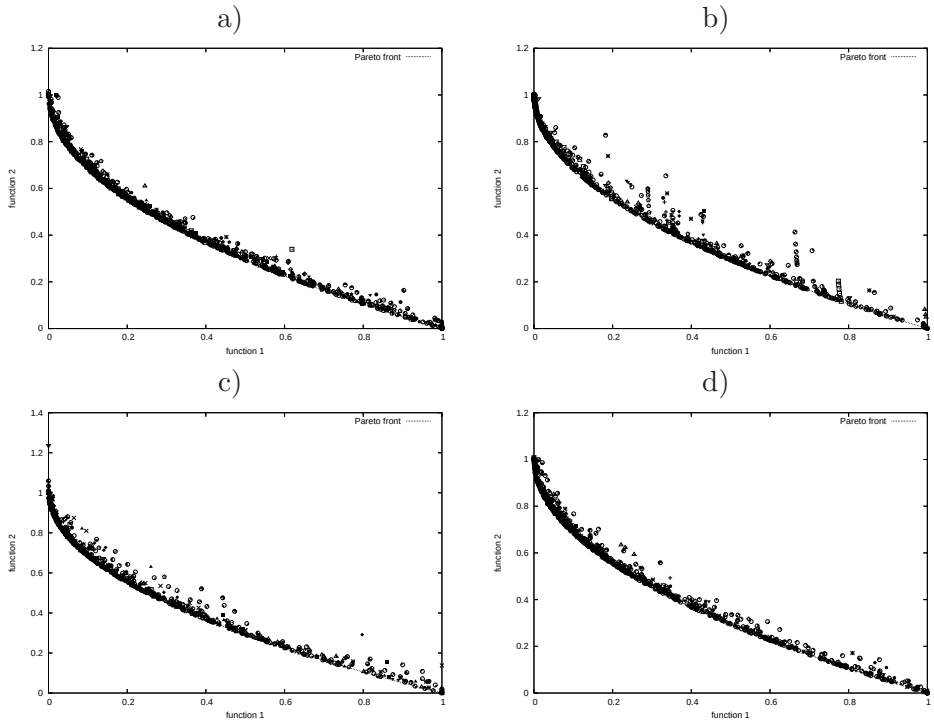

Figure A.11: Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for Deb's test function.

Figure A.11 shows the graphical results produced by each gbest selection strategy in Deb1's test function. The true Pareto front of the problem is shown as a continuous line. Figure A.12 shows the statistics previously described for this test function.

A.7 Deb2's test function

This test function is described in Section 2.5.2.

Figure A.13 shows the graphical results produced by each gbest selection strategy in Deb2's test function. The true Pareto front of the problem is shown as a continuous line. Figure A.14 shows the statistics previously described for this test function.

A.8 Scheme 4

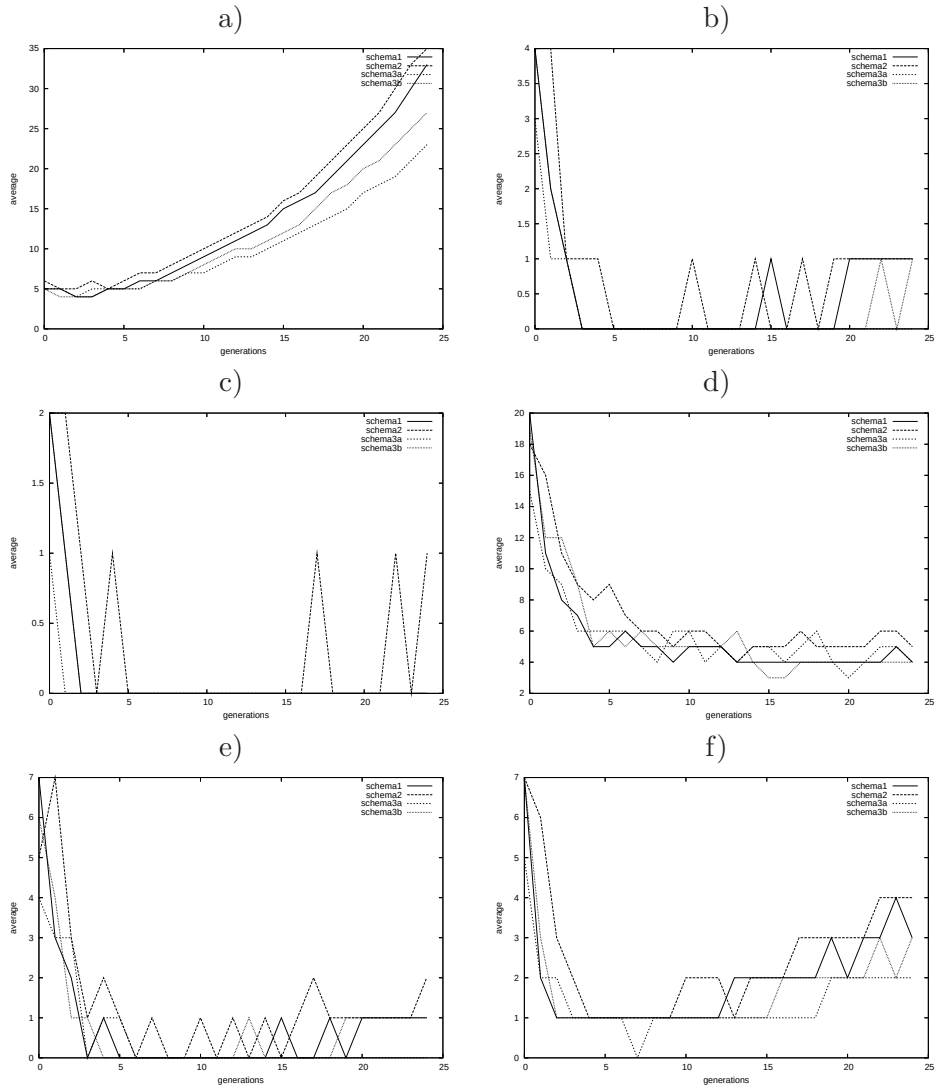

Figure A.12: Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for Deb's test function.

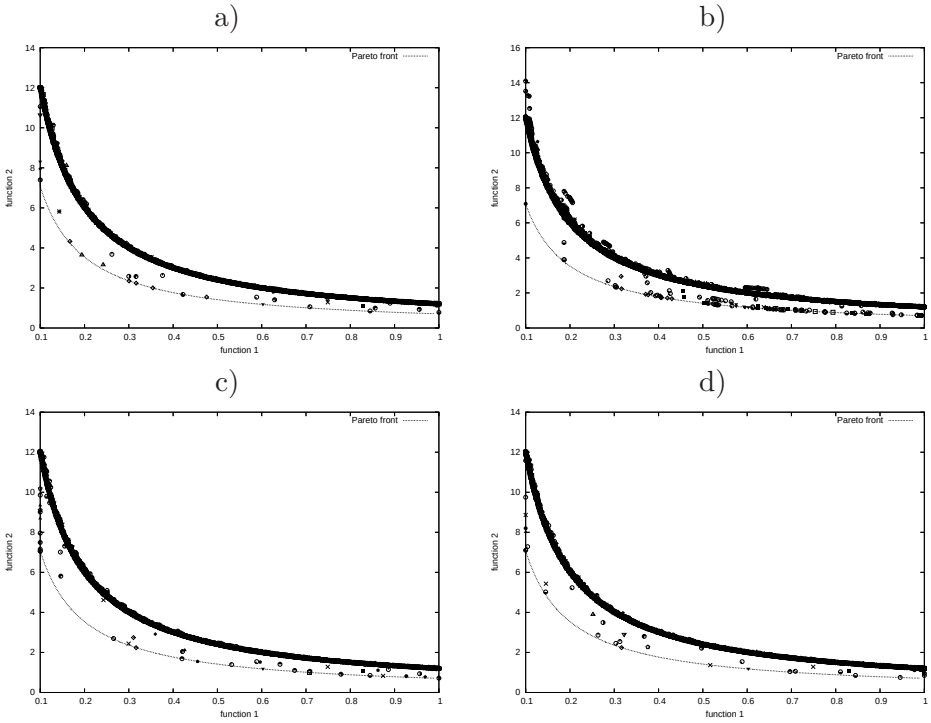

Figure A.13: Pareto fronts produced by schemes a) 1, b) 2, c) 3a and d) 3b for Deb2's test function.

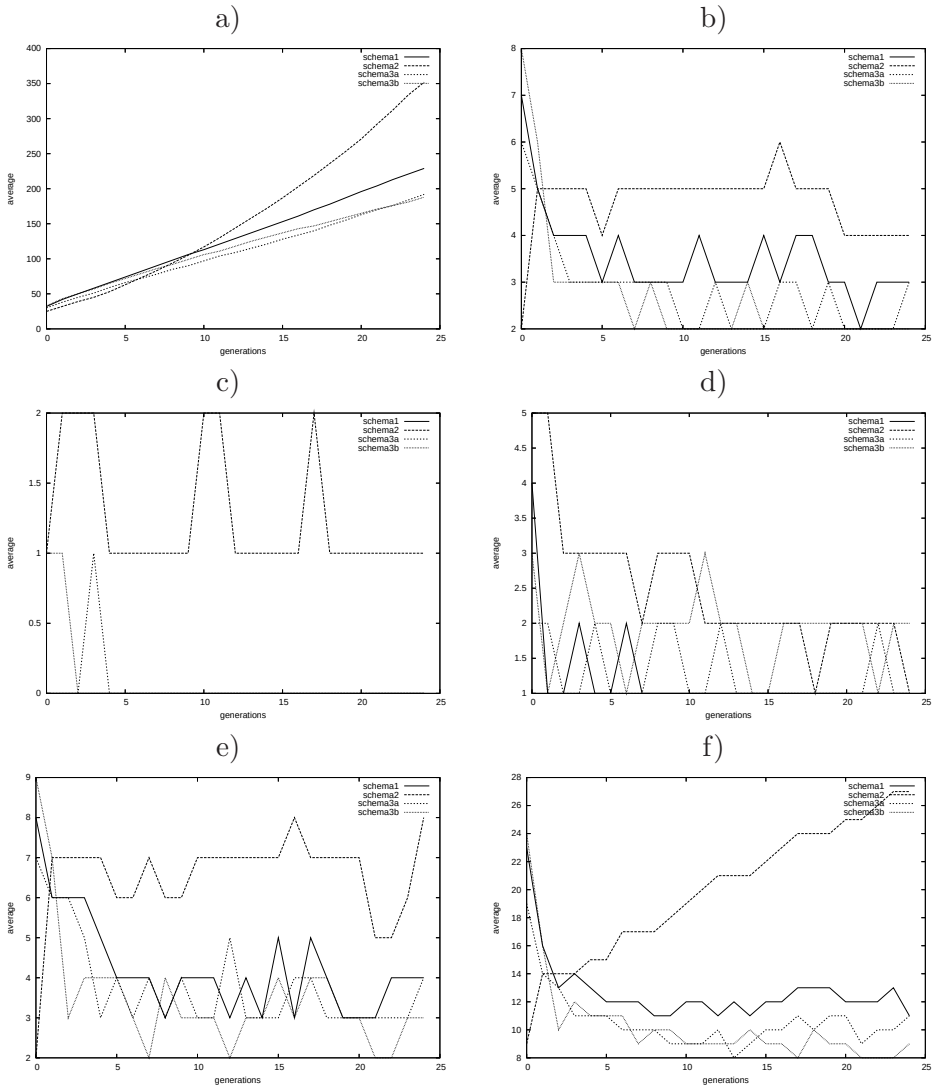

Figure A.14: Plots that indicate a) the total of non-dominated solutions, b) the occurrences of strong dominance, c) total of replacements of a leader by the new particle's position, d) total of pbest replacements, e) the total of particles which were dominated and f) the quantity of new non-dominated particles, for DEB2's test function.

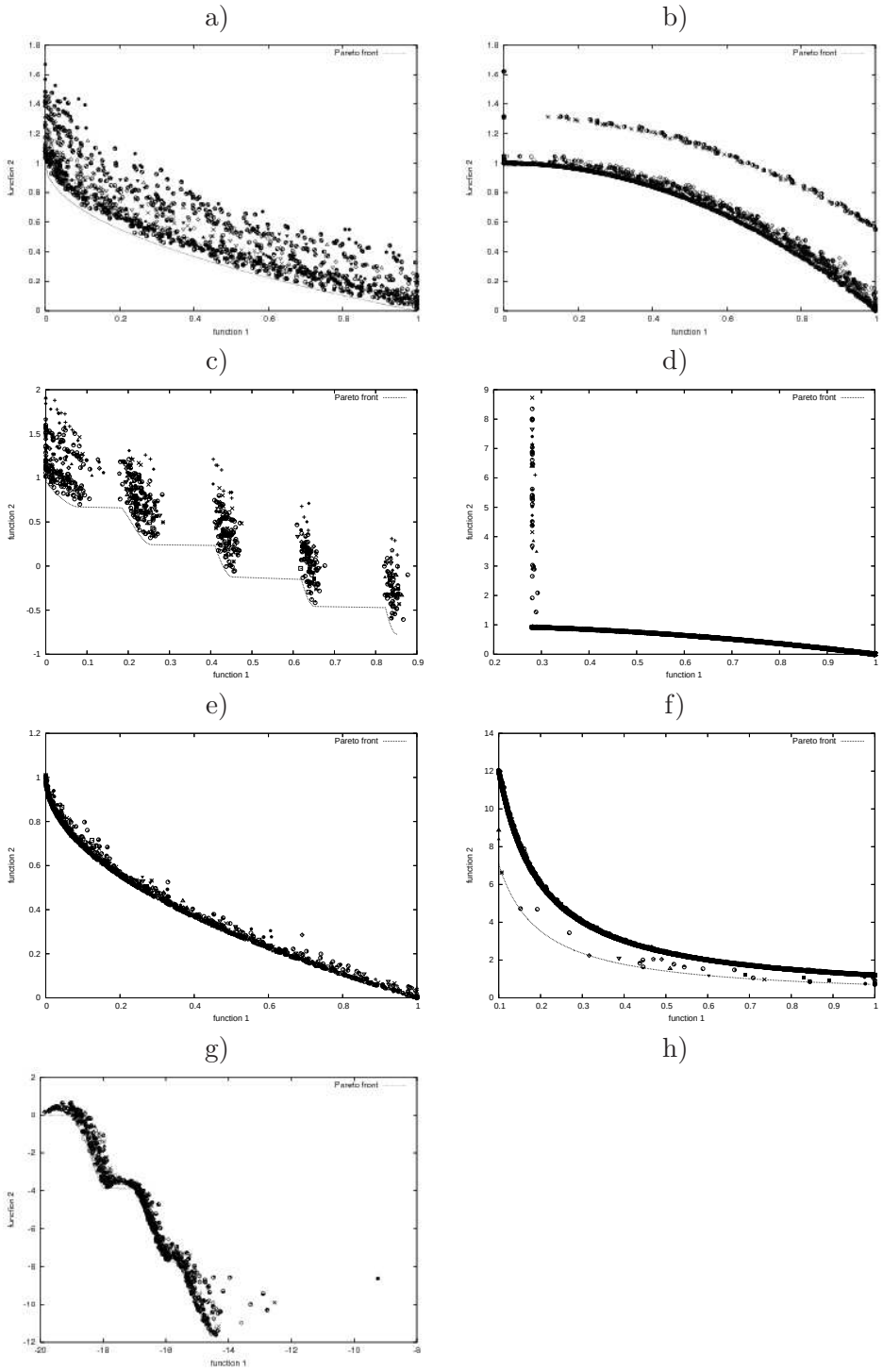

Figure A.15: Pareto fronts produced by the scheme 4 for test functions

B

B.1 Experiment 1

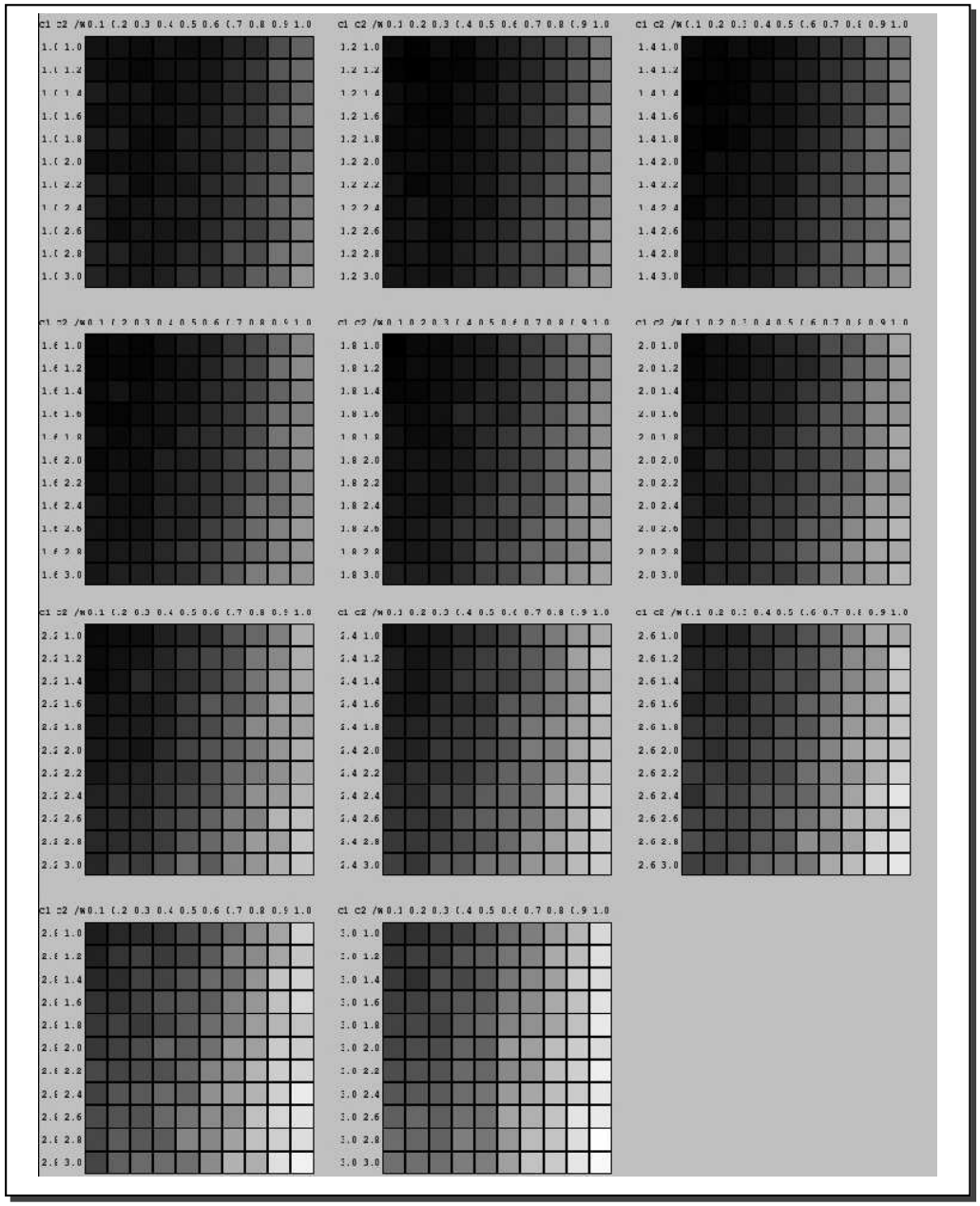

Figure B.1: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kursawe's test function.

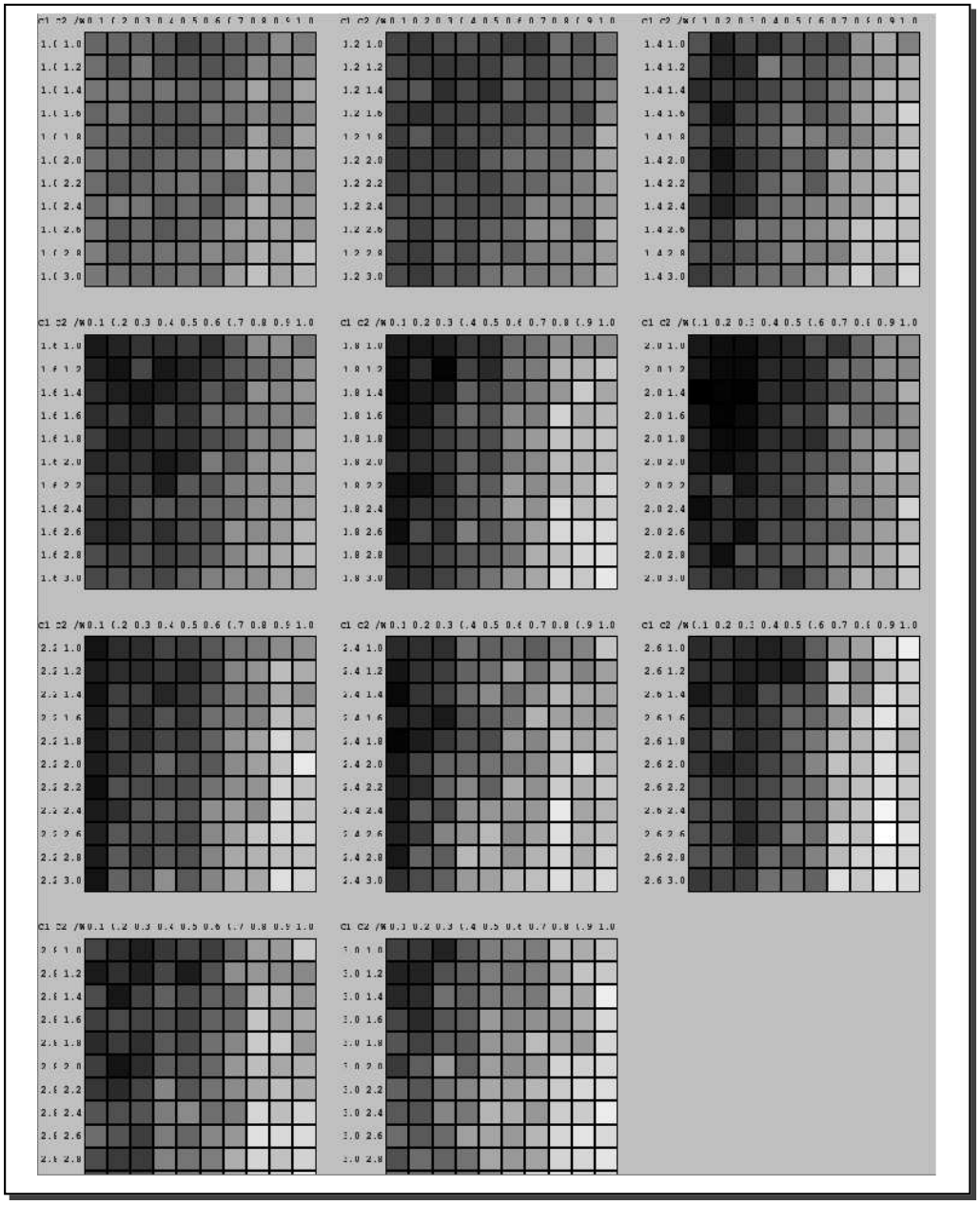

Figure B.2: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb1's test function.

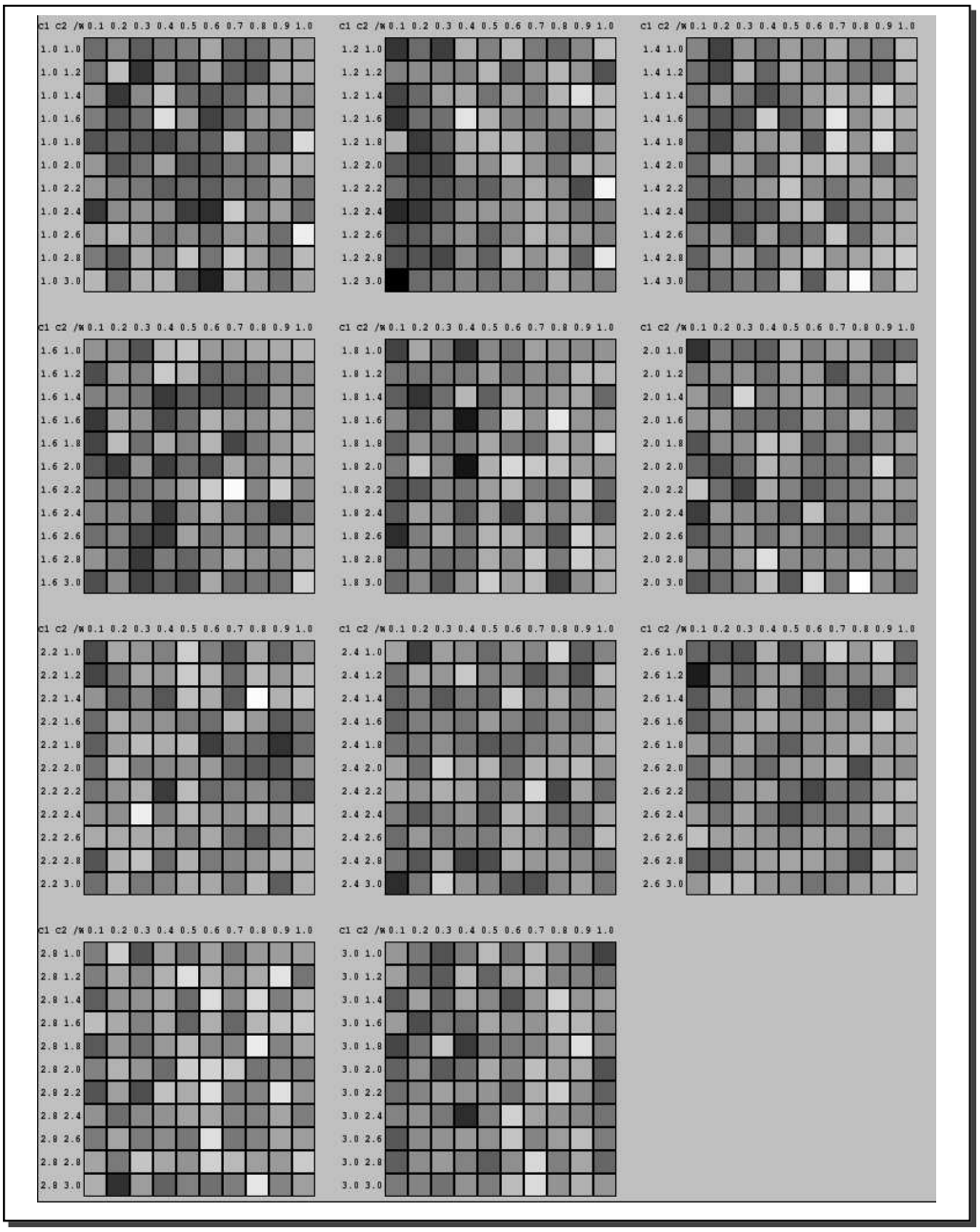

Figure B.3: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb2's test function.

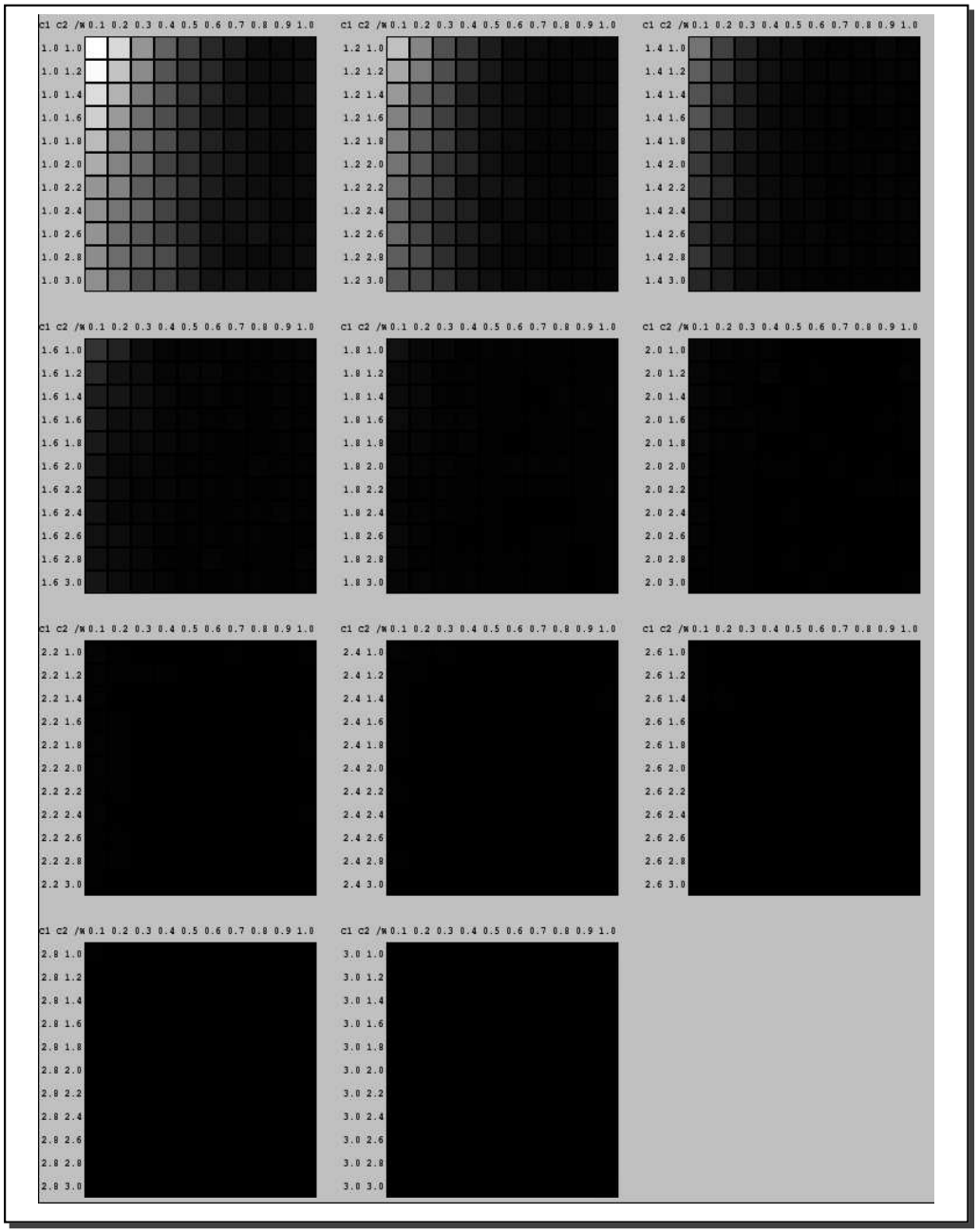

Figure B.4: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT1's test function.

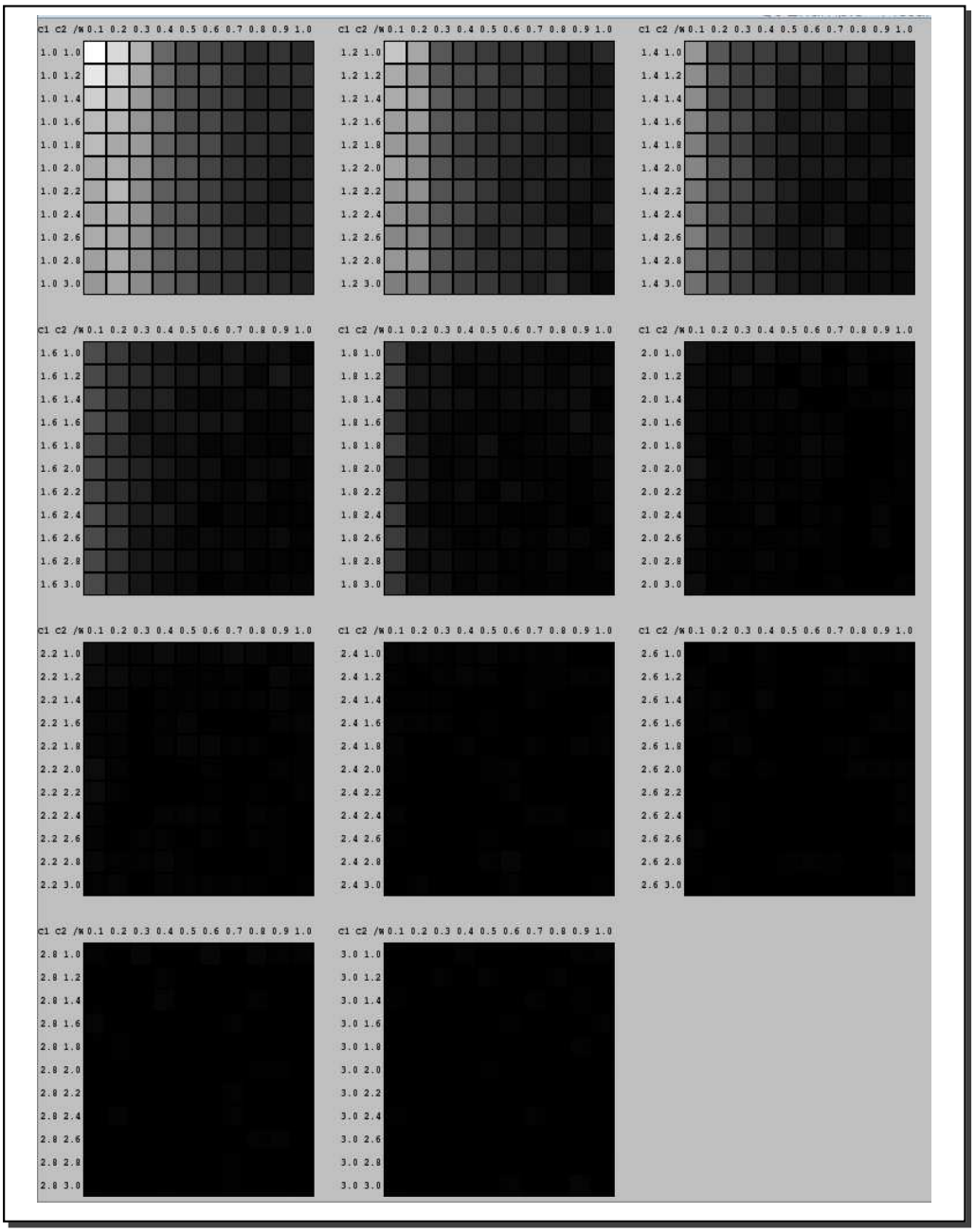

Figure B.5: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT2's test function.

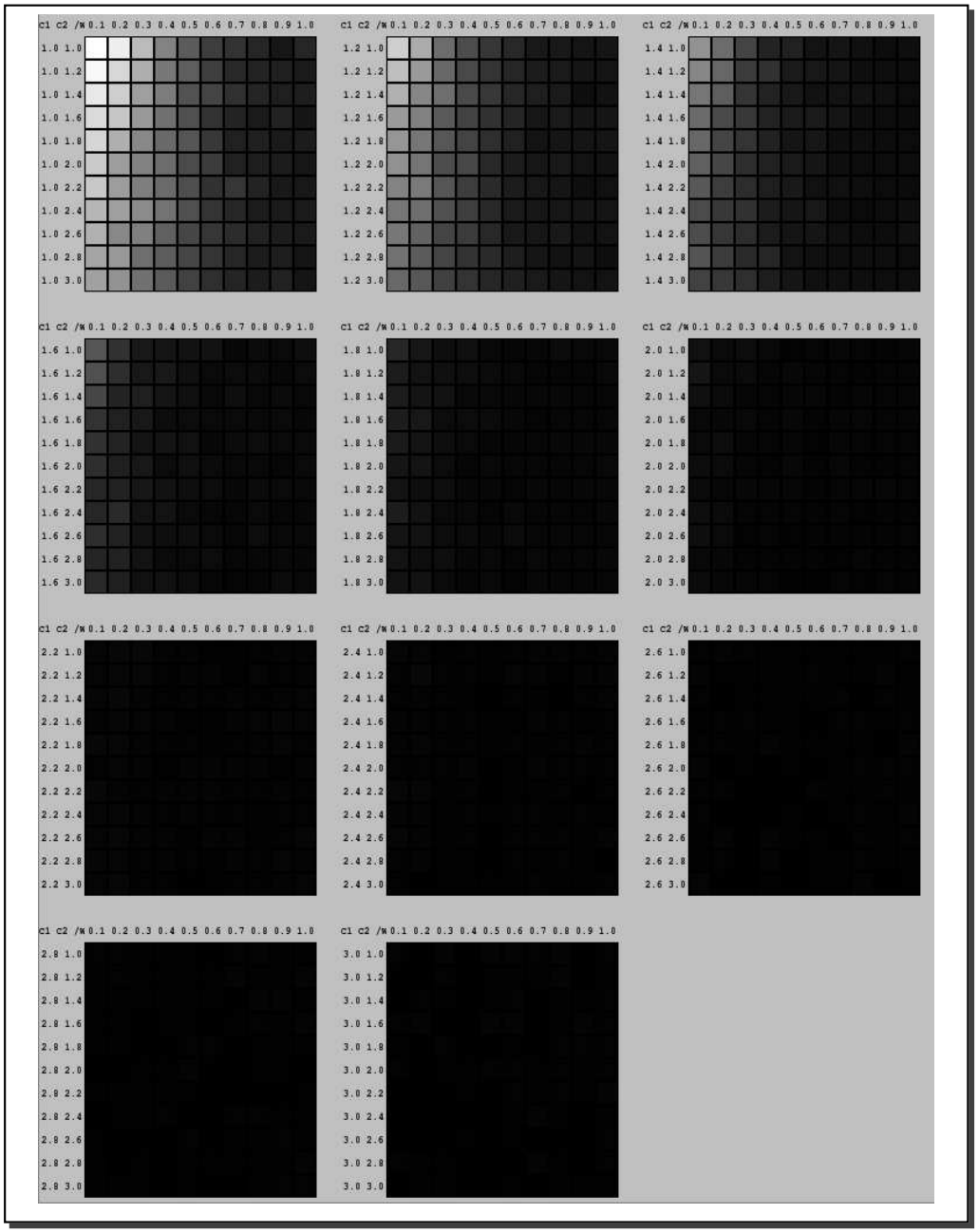

Figure B.6: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT3's test function.

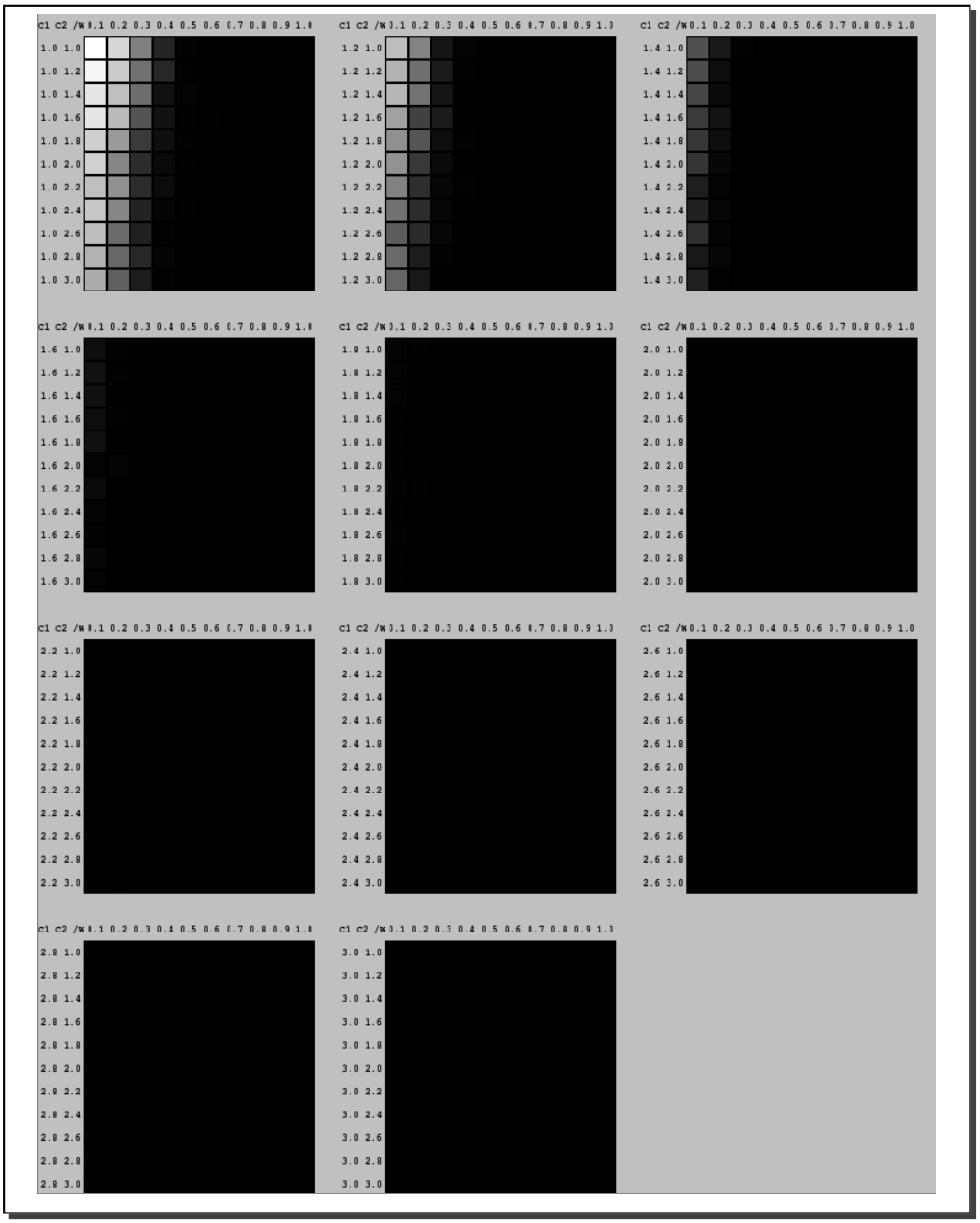

Figure B.7: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT6's test function.

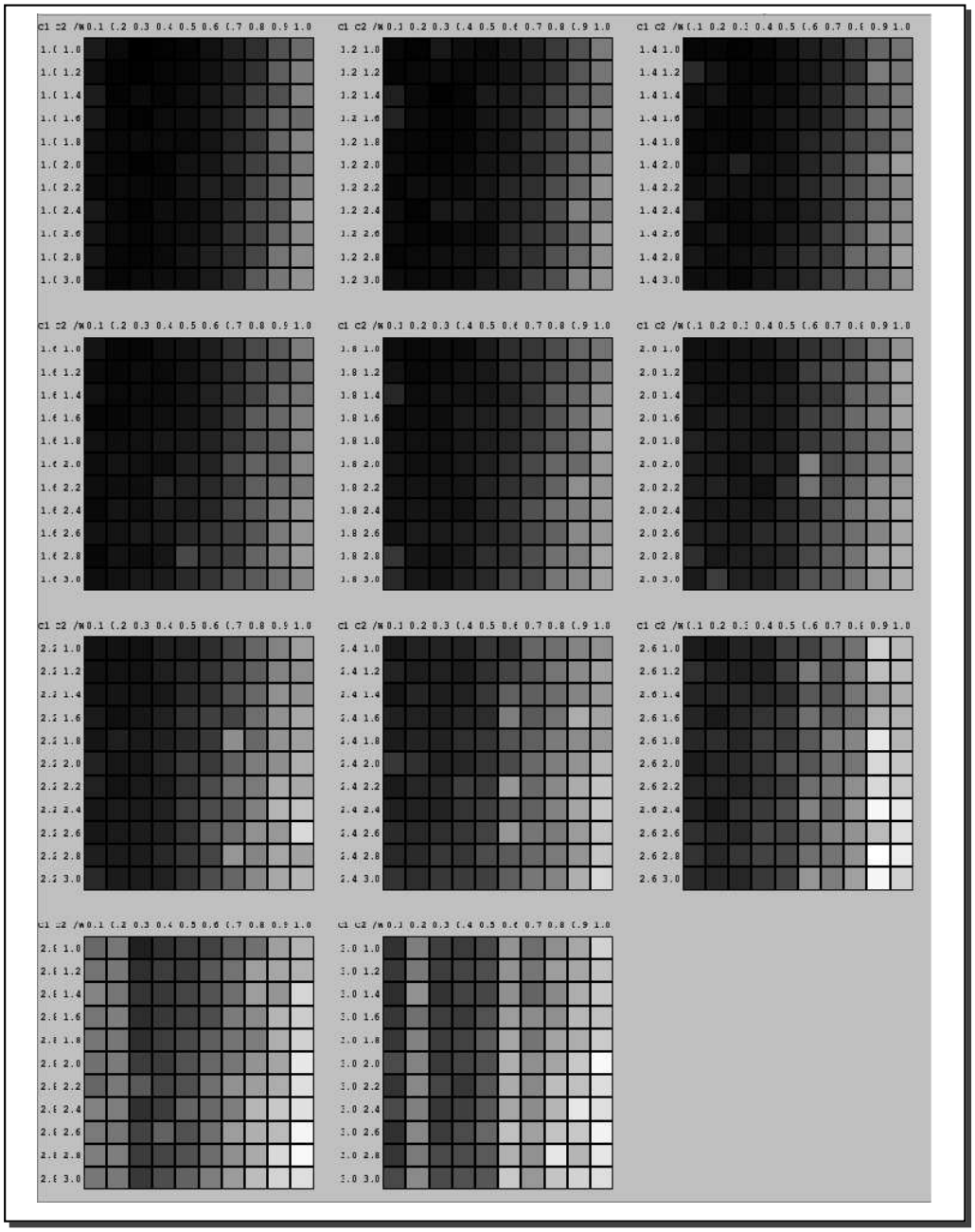

Figure B.8: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kita's test function.

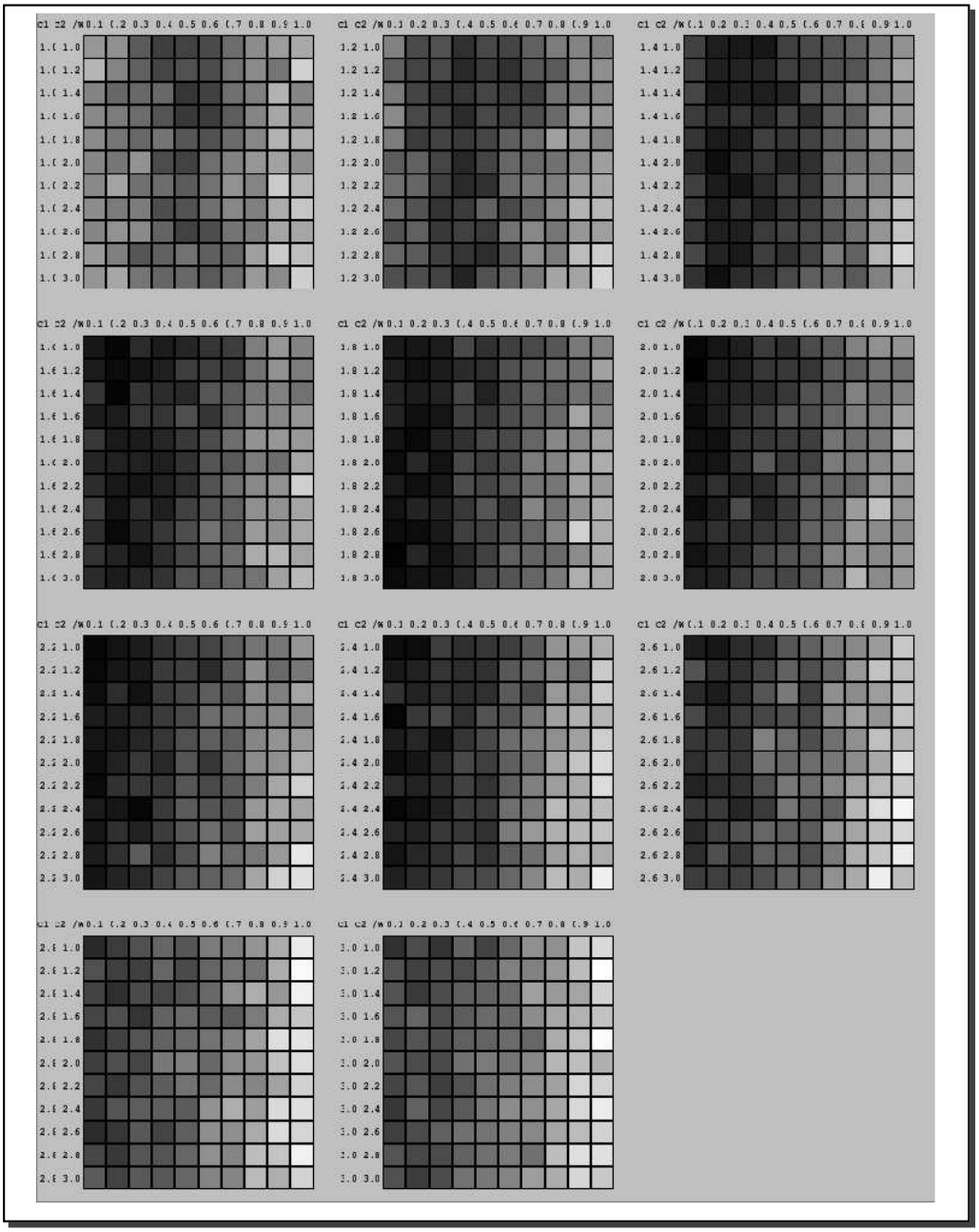

Figure B.9: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Welded beam test function.

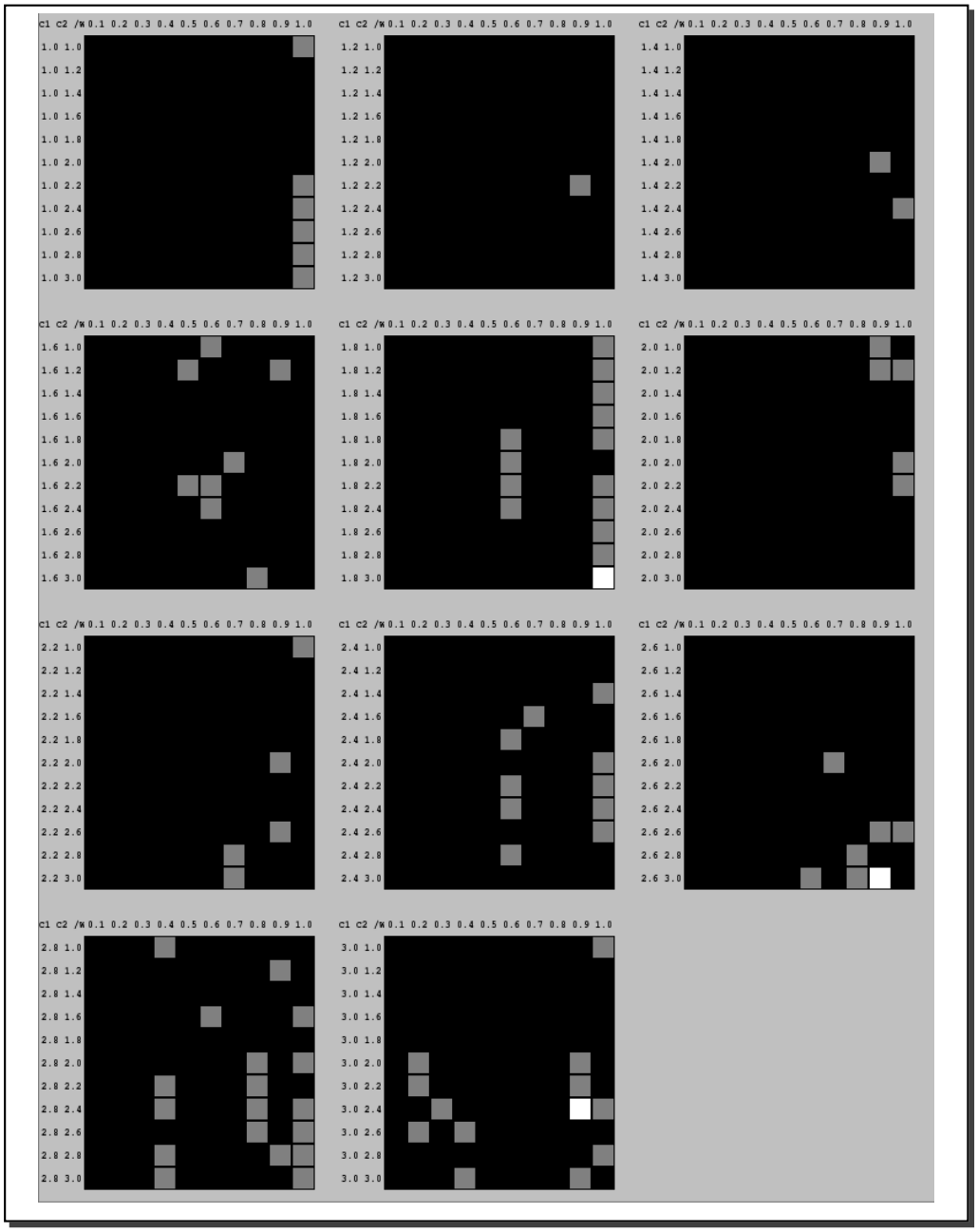

Figure B.10: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Osyczka2's test function.

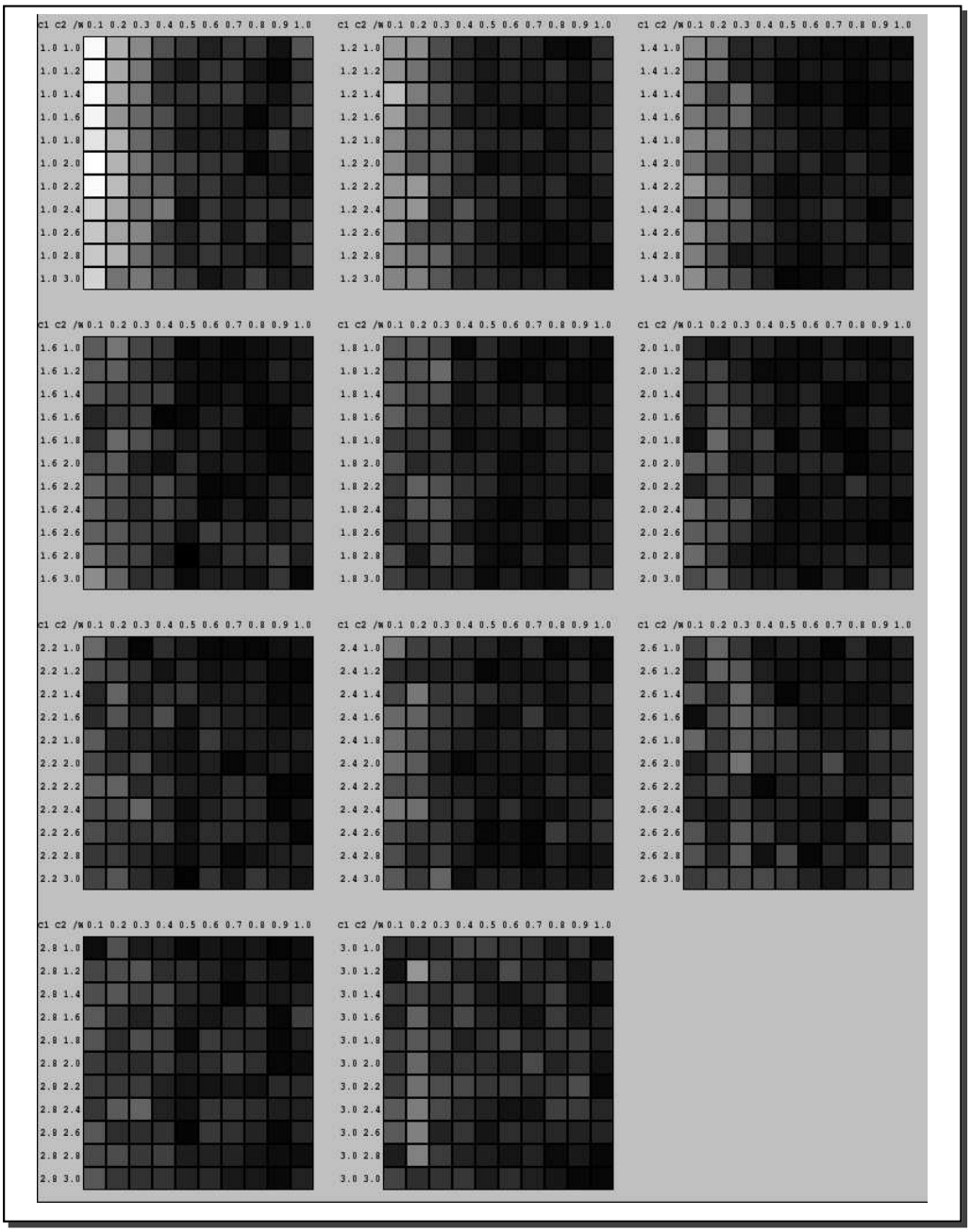

Figure B.11: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Speed Reducer test function.

B.2 Experiment 2

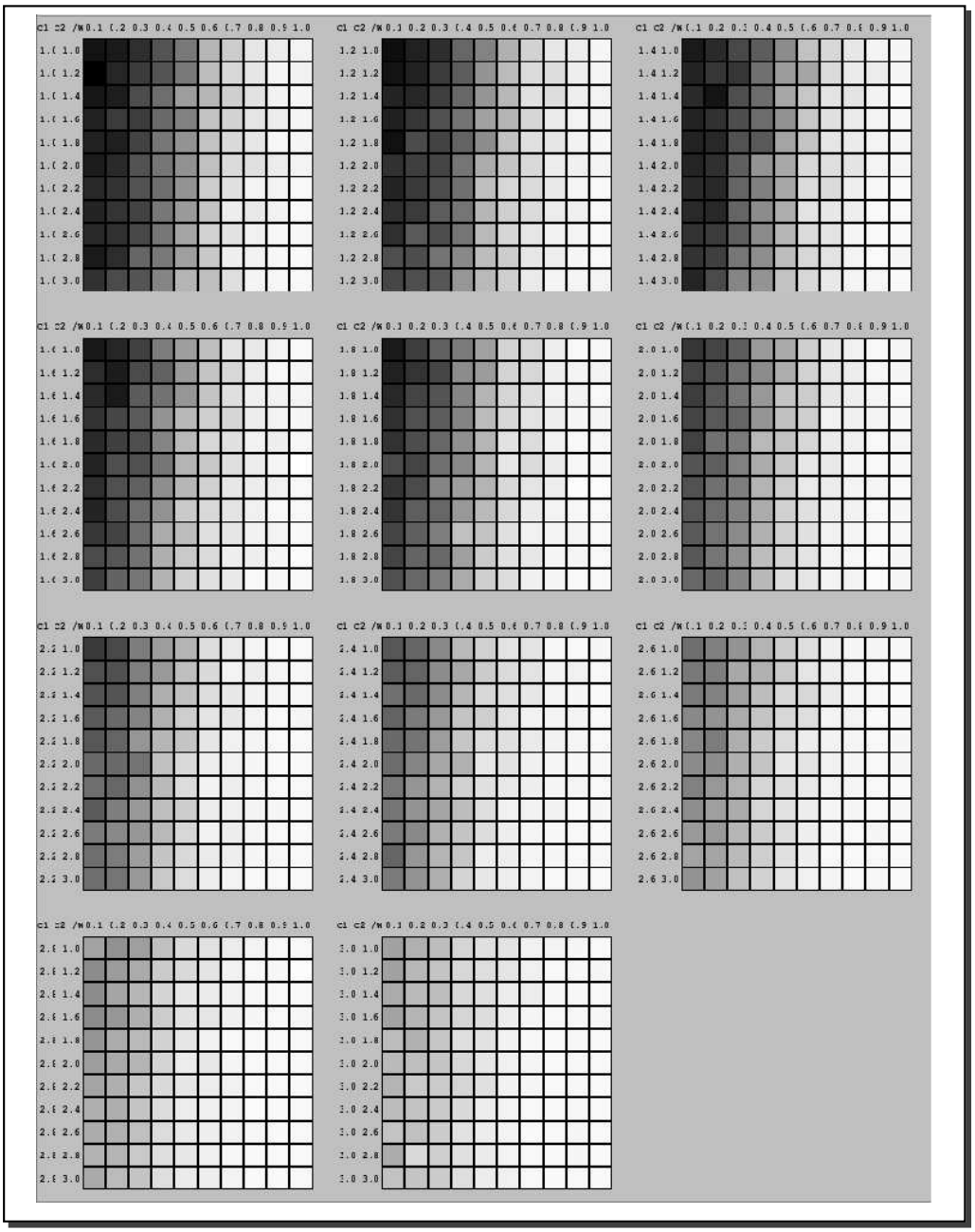

Figure B.12: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kursawe's test function.

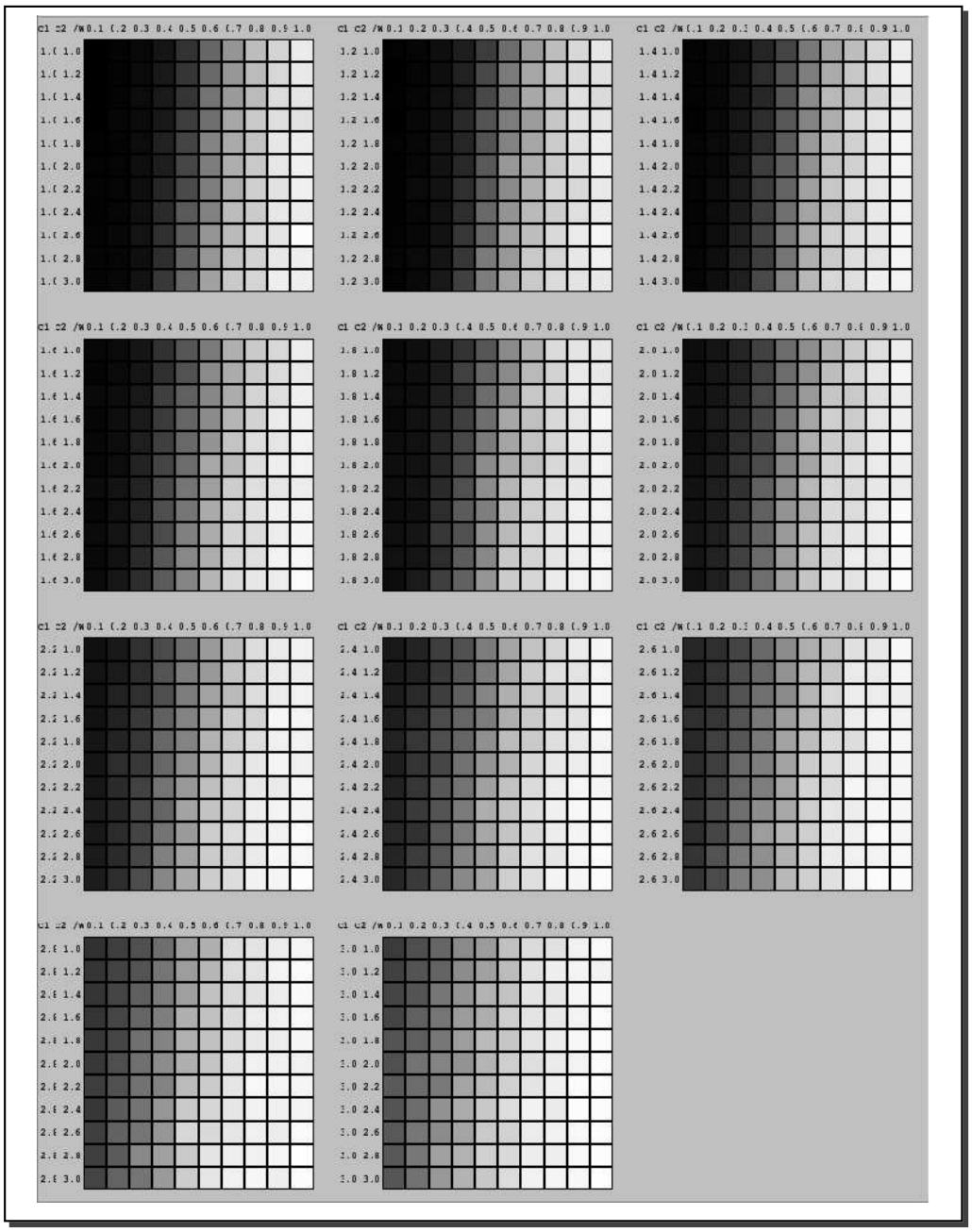

Figure B.13: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb1's test function.

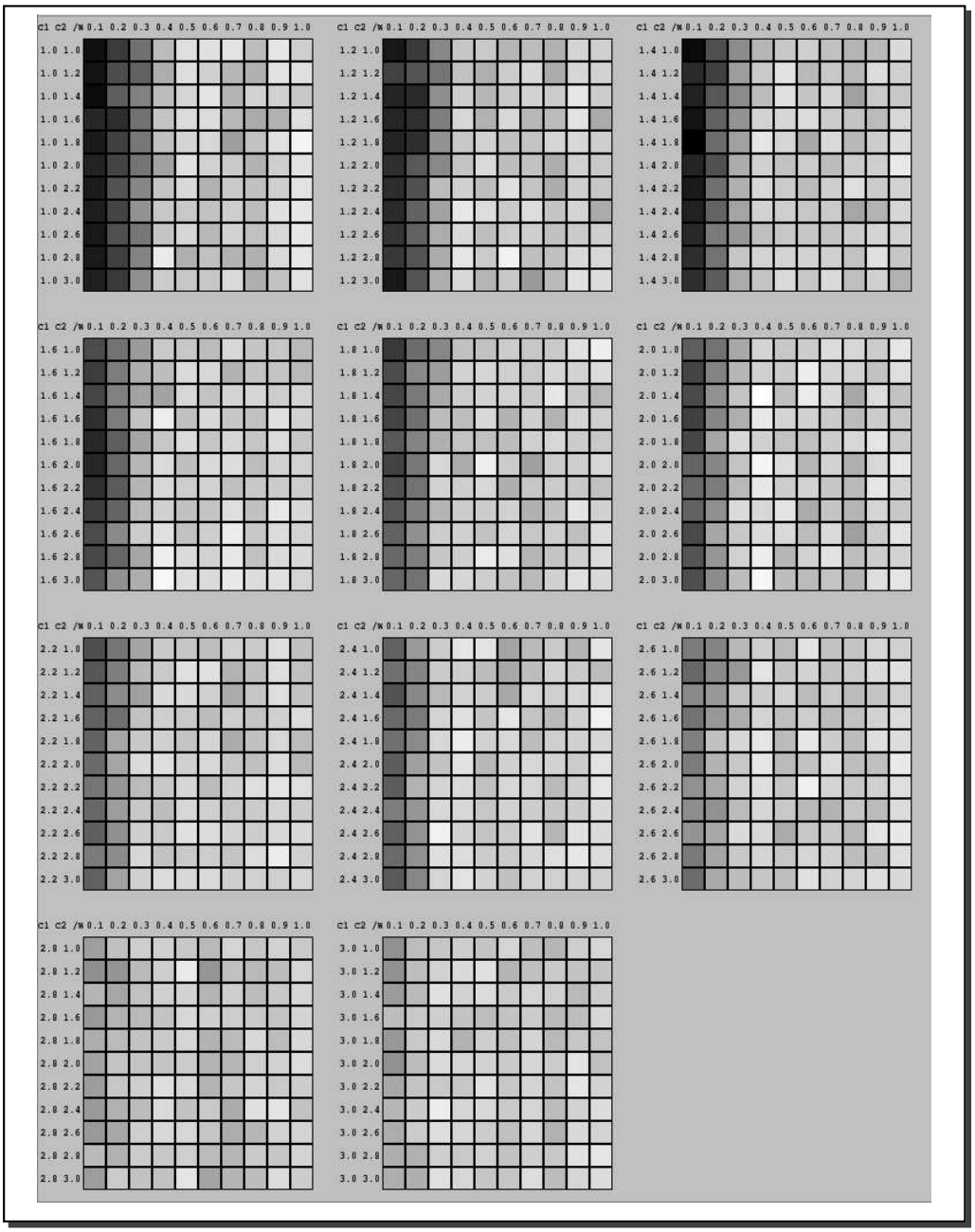

Figure B.14: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb2's test function.

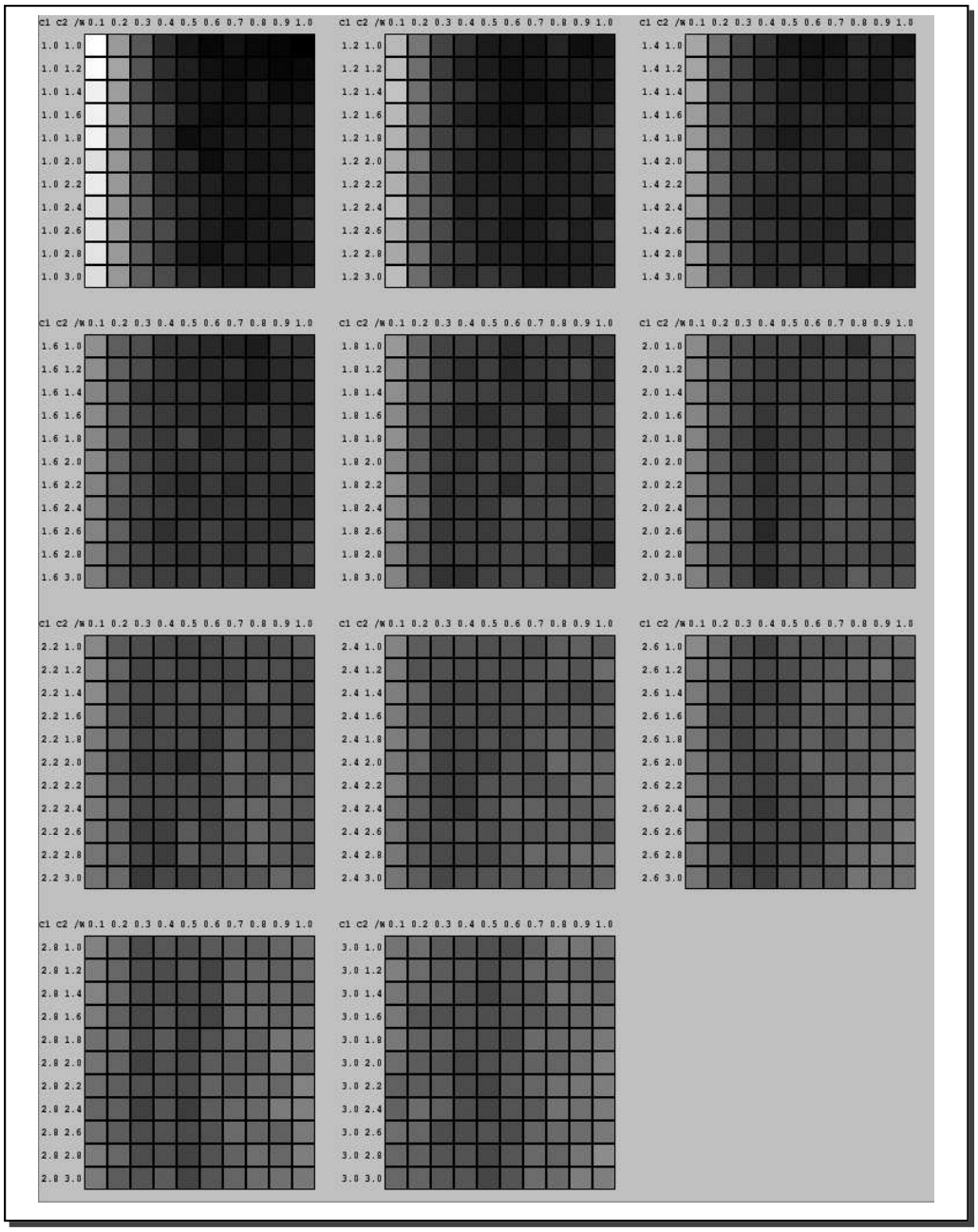

Figure B.15: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT1's test function.

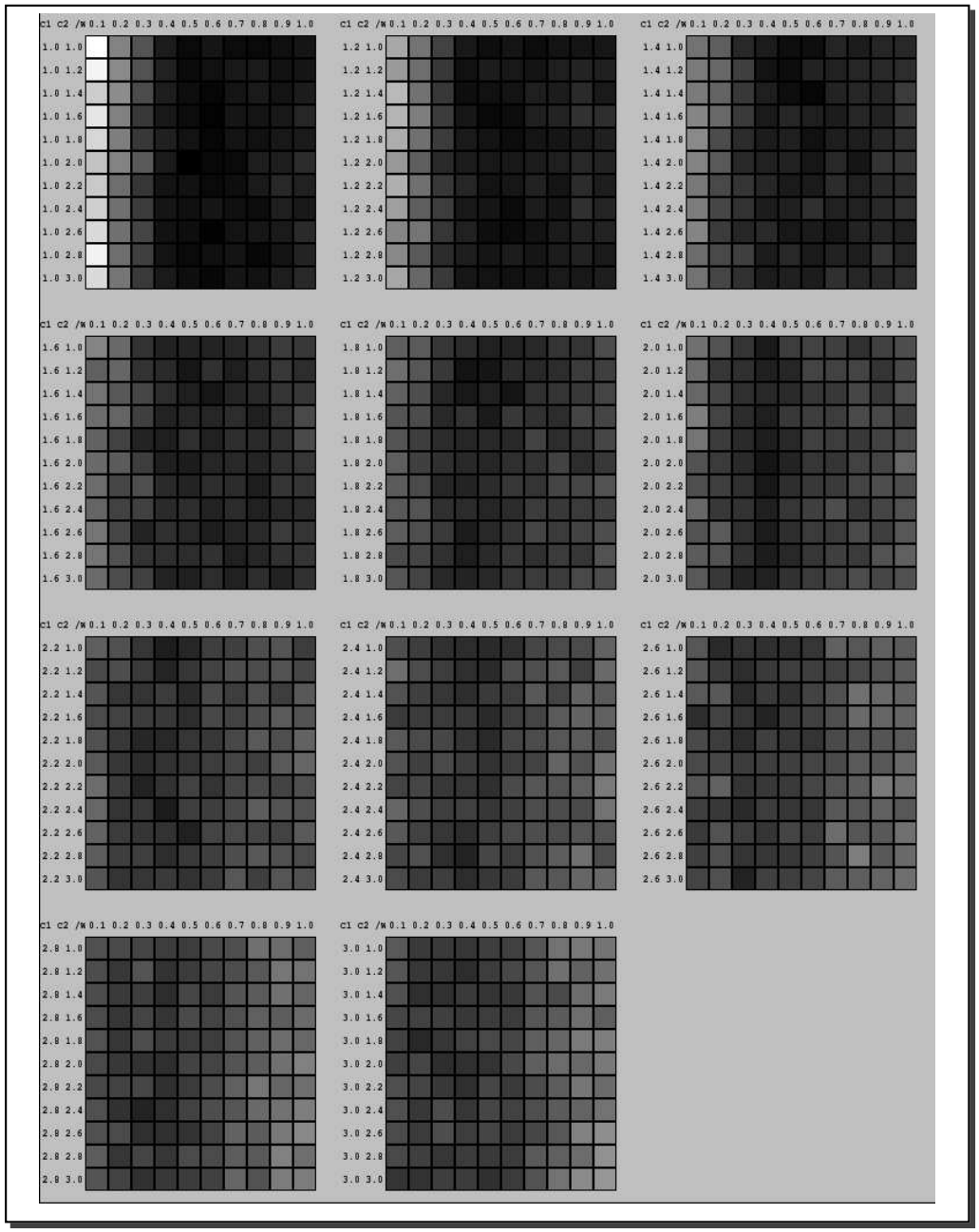

Figure B.16: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT2's test function.

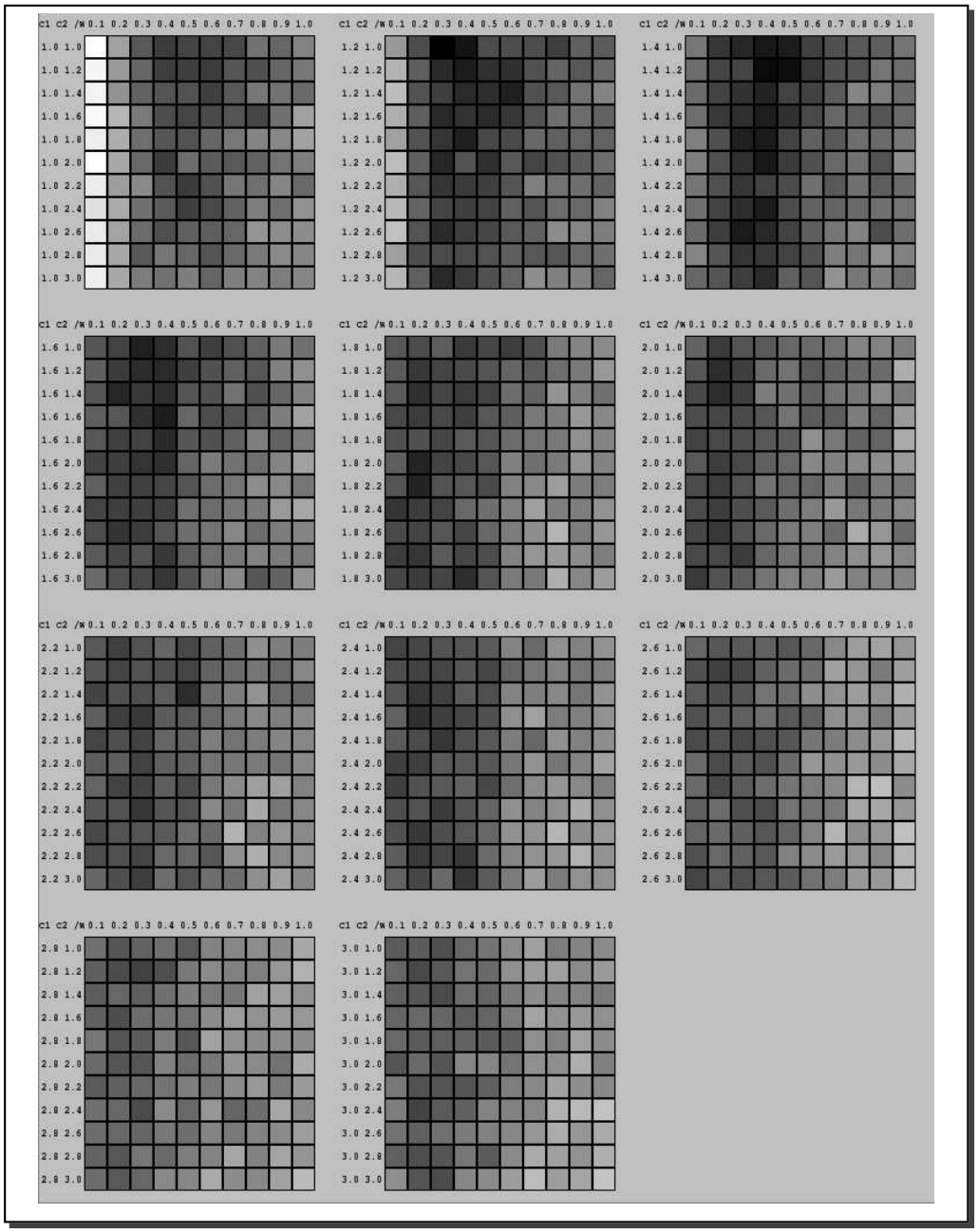

Figure B.17: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT3's test function.

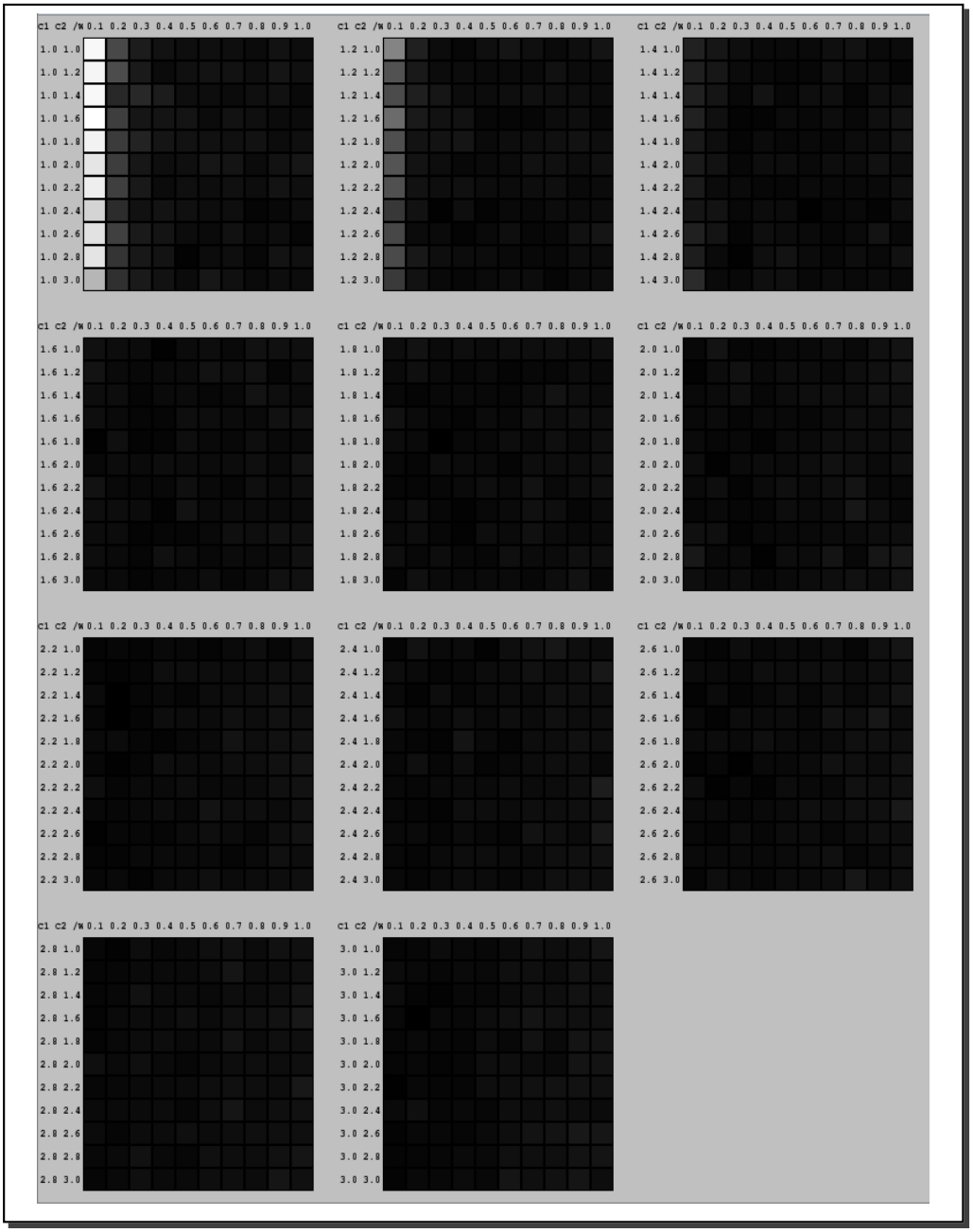

Figure B.18: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT6's test function.

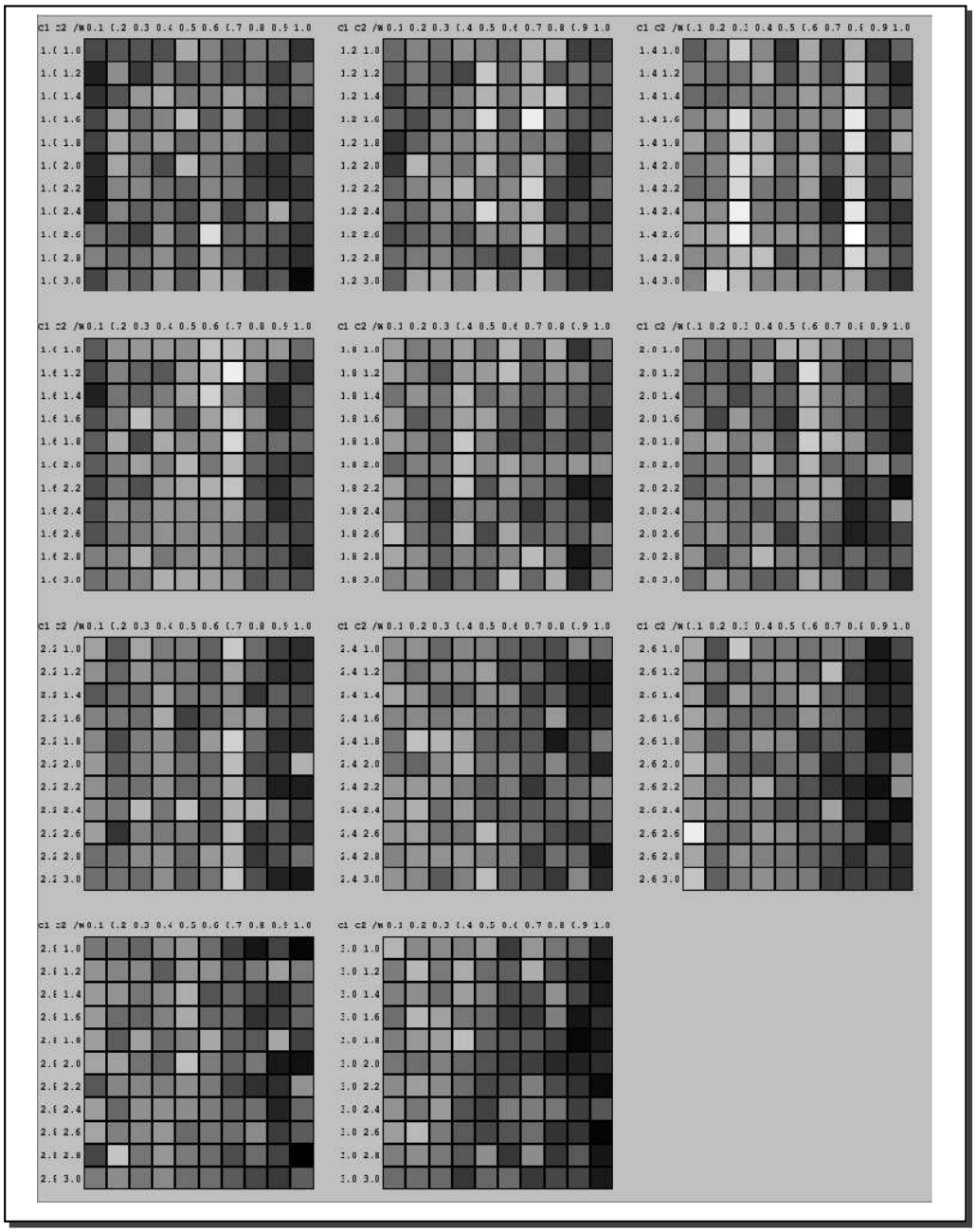

Figure B.19: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kita's test function.

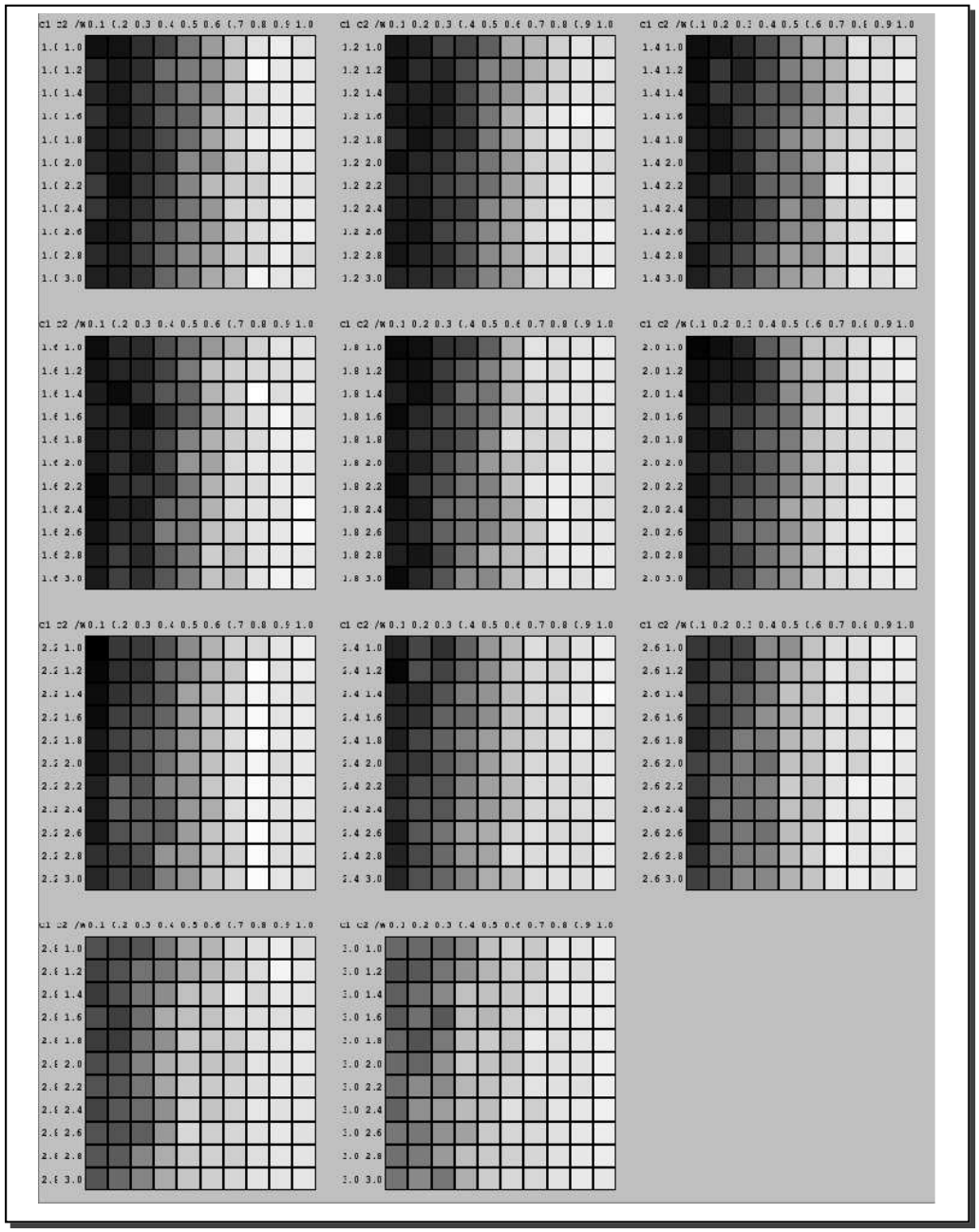

Figure B.20: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Welded Beam test function.

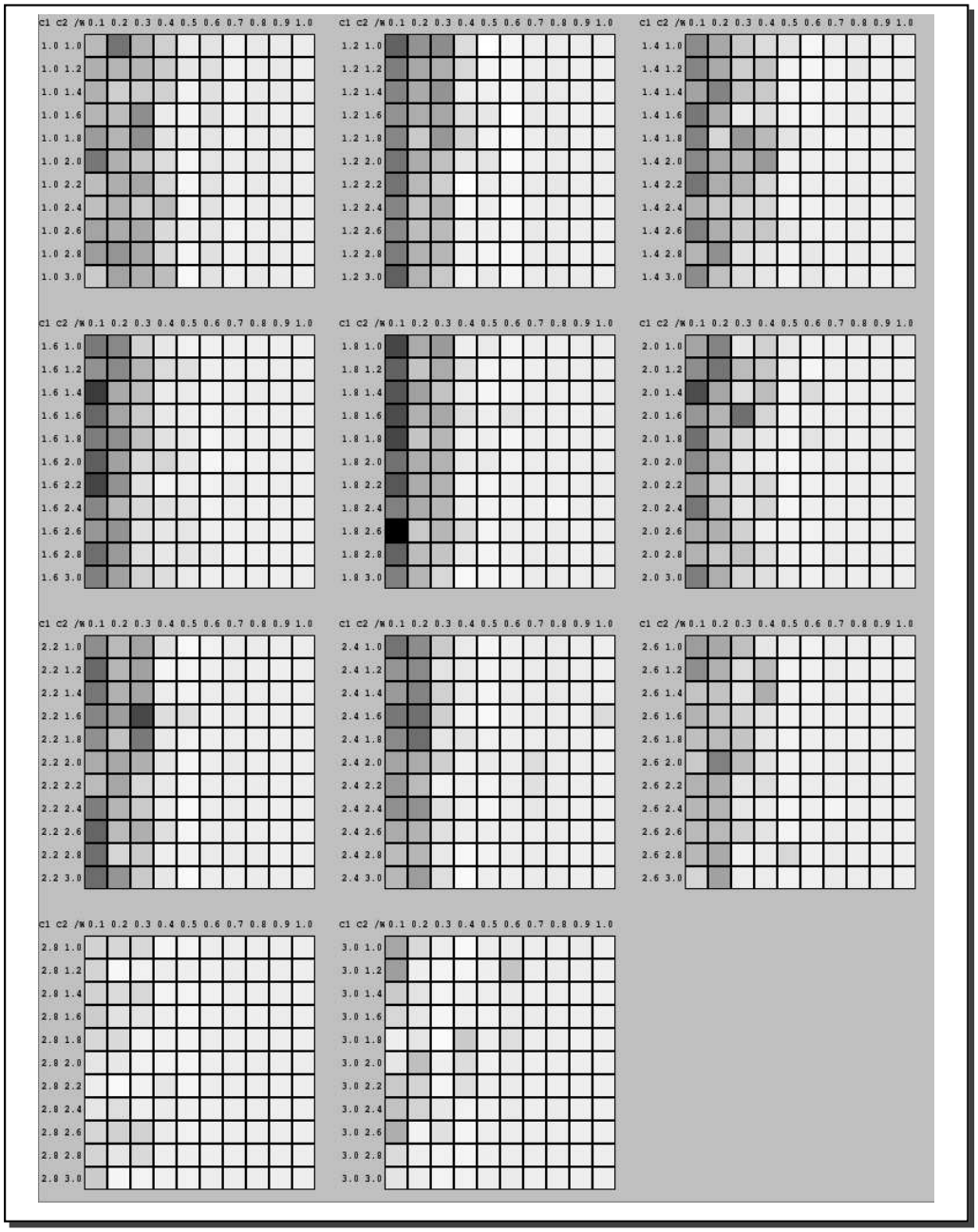

Figure B.21: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Osyczka2's test function.

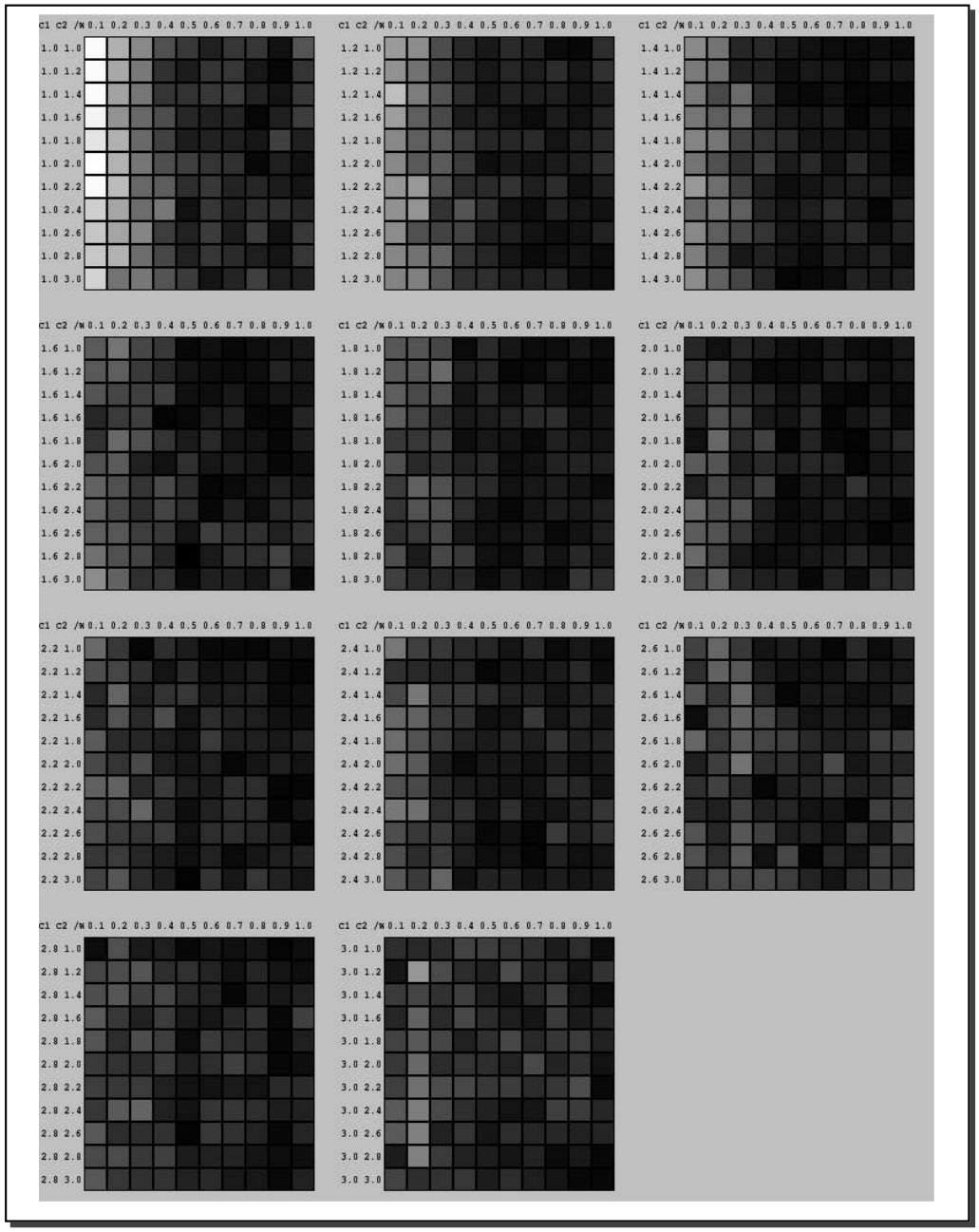

Figure B.22: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the the Speed Reducer test function.

B.3 Experiment 3

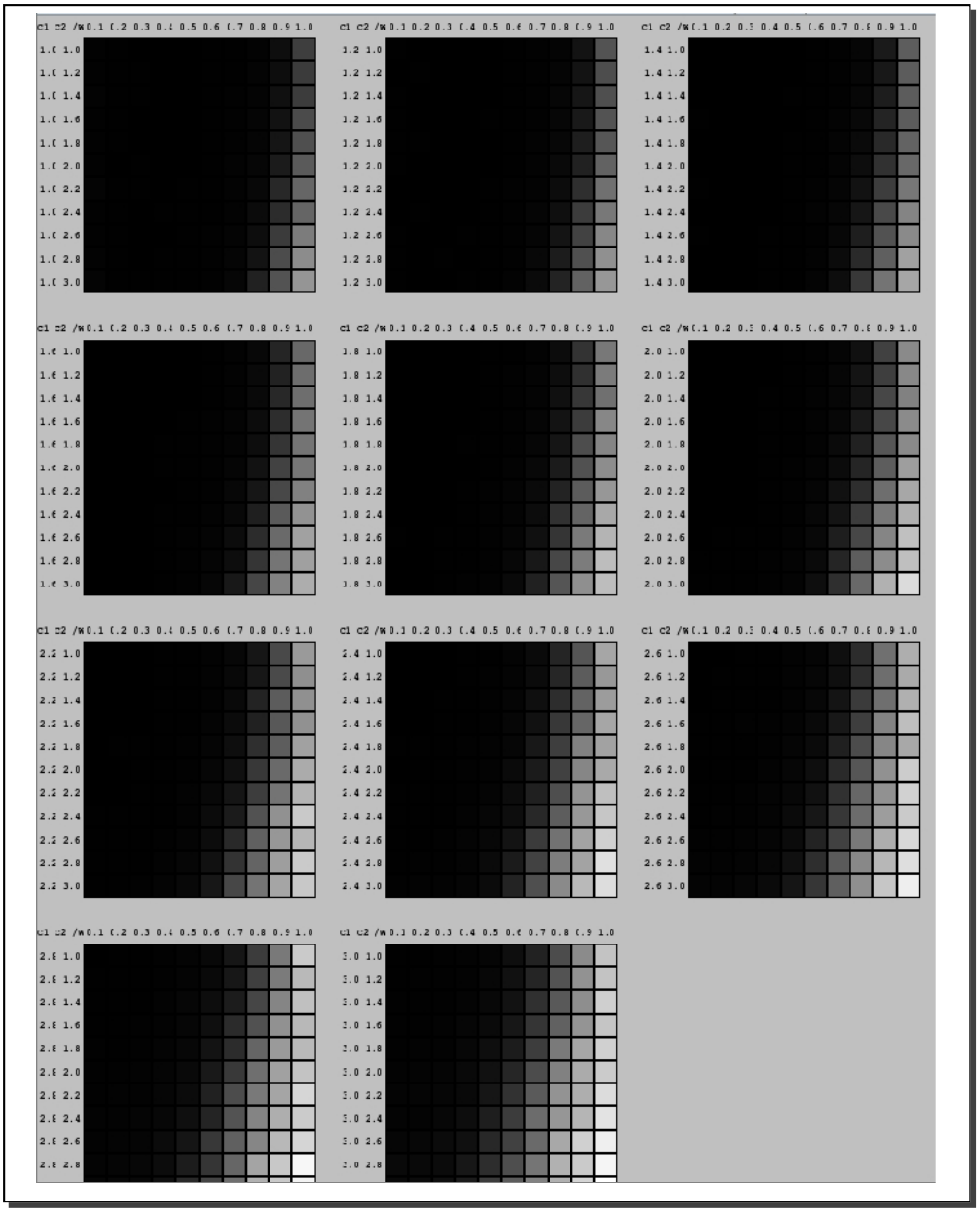

Figure B.23: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kursawe's test function.

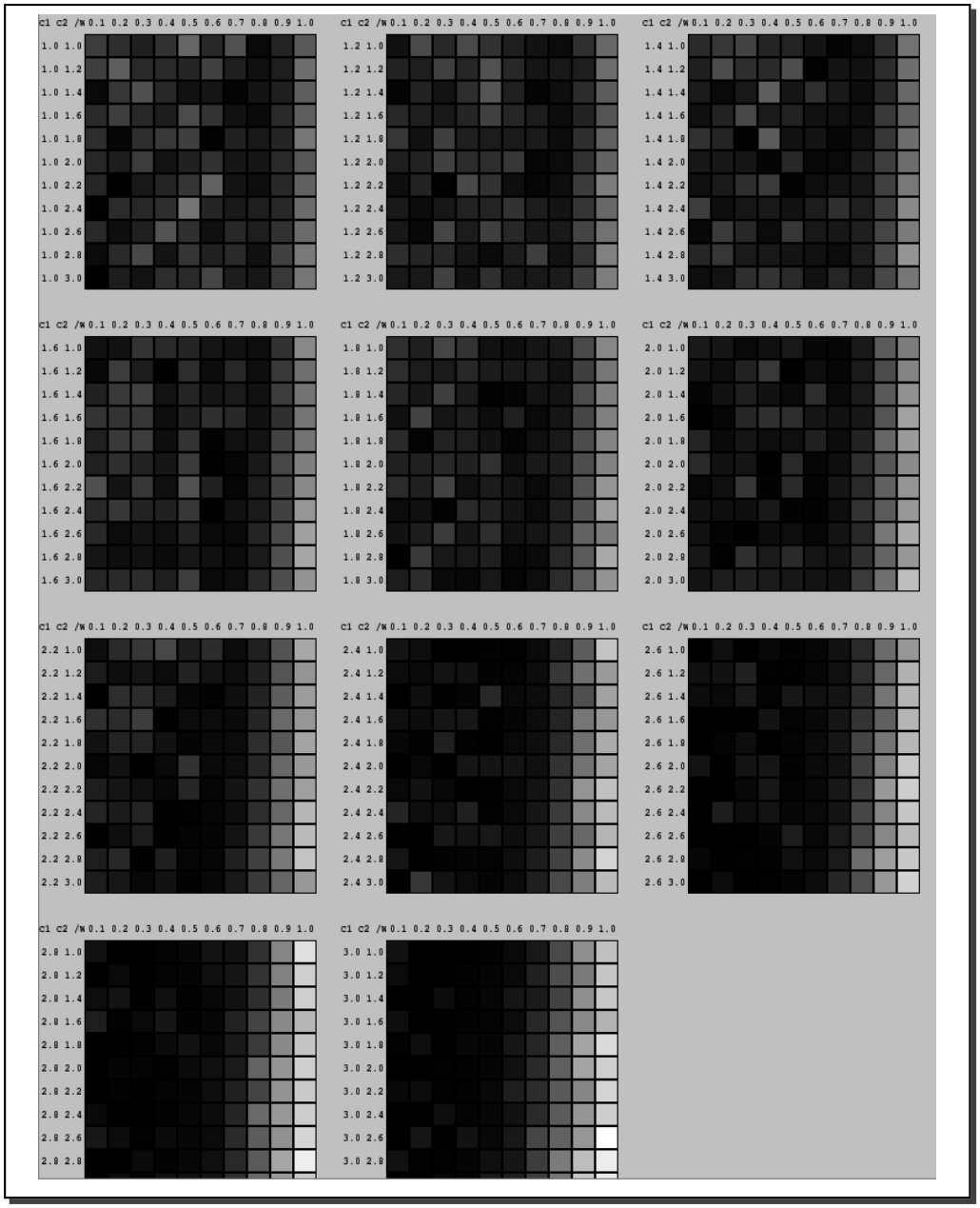

Figure B.24: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Deb2's test function.

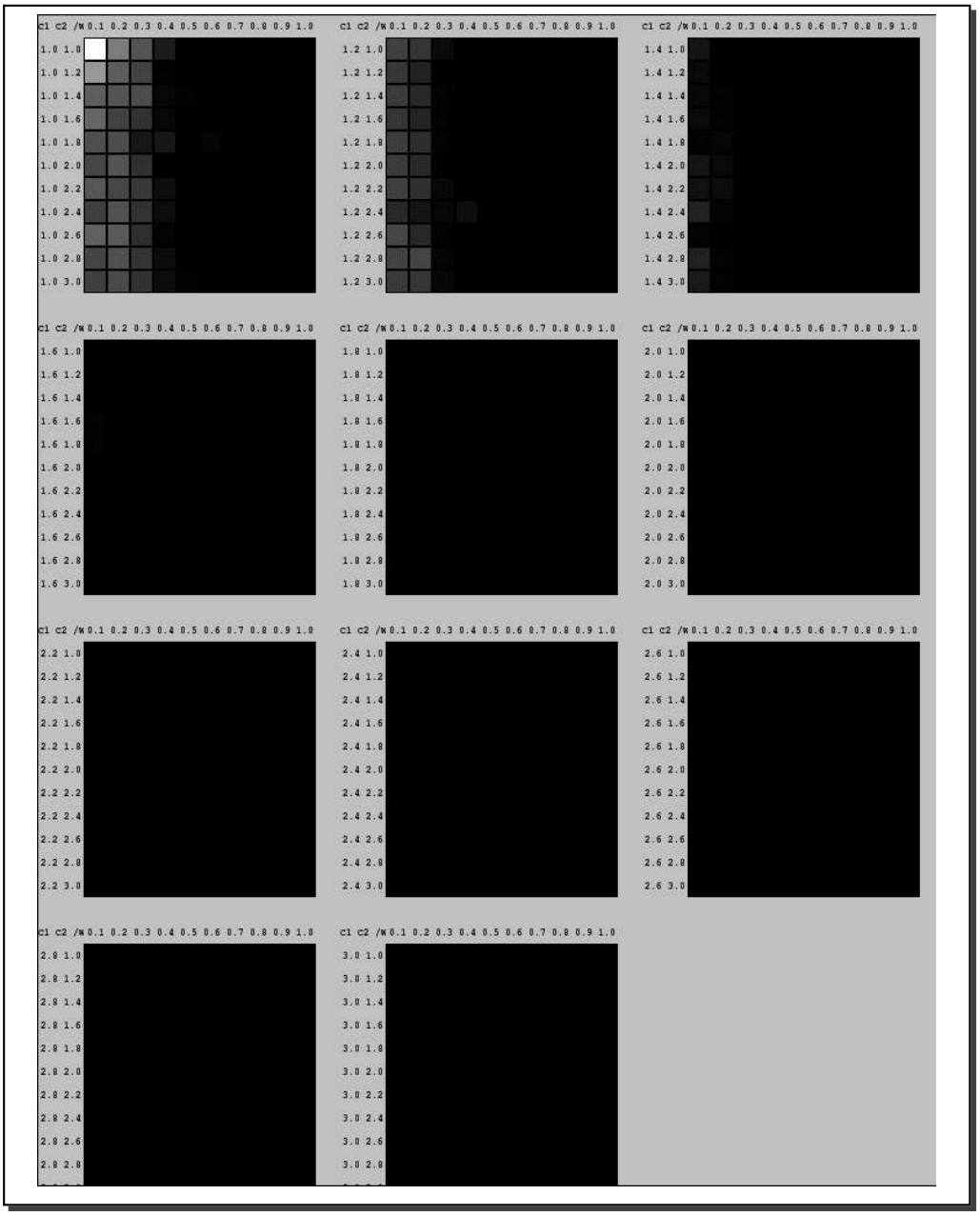

Figure B.25: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of ZDT3's test function.

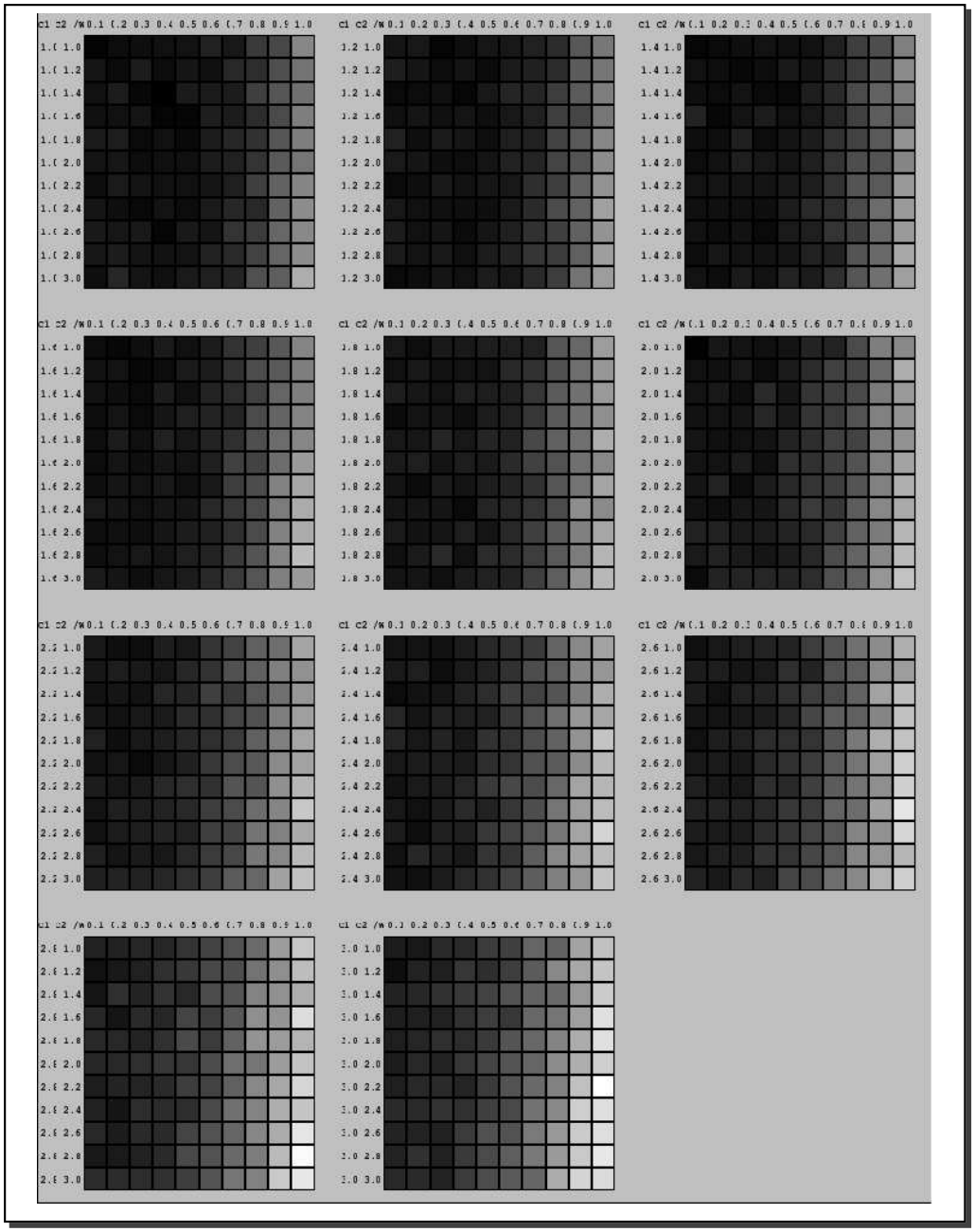

Figure B.26: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Kita's test function.

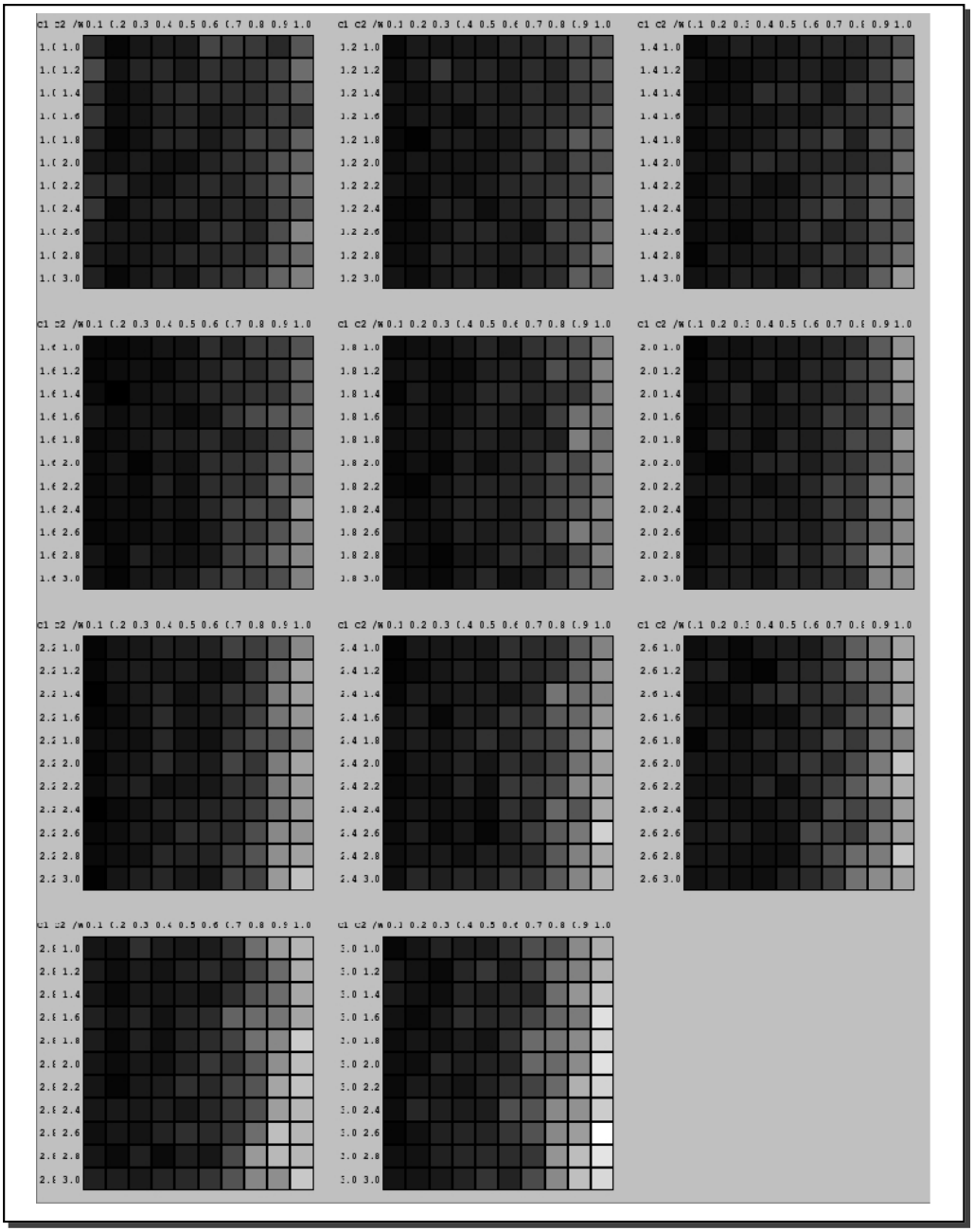

Figure B.27: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of the Welded Beam test function.

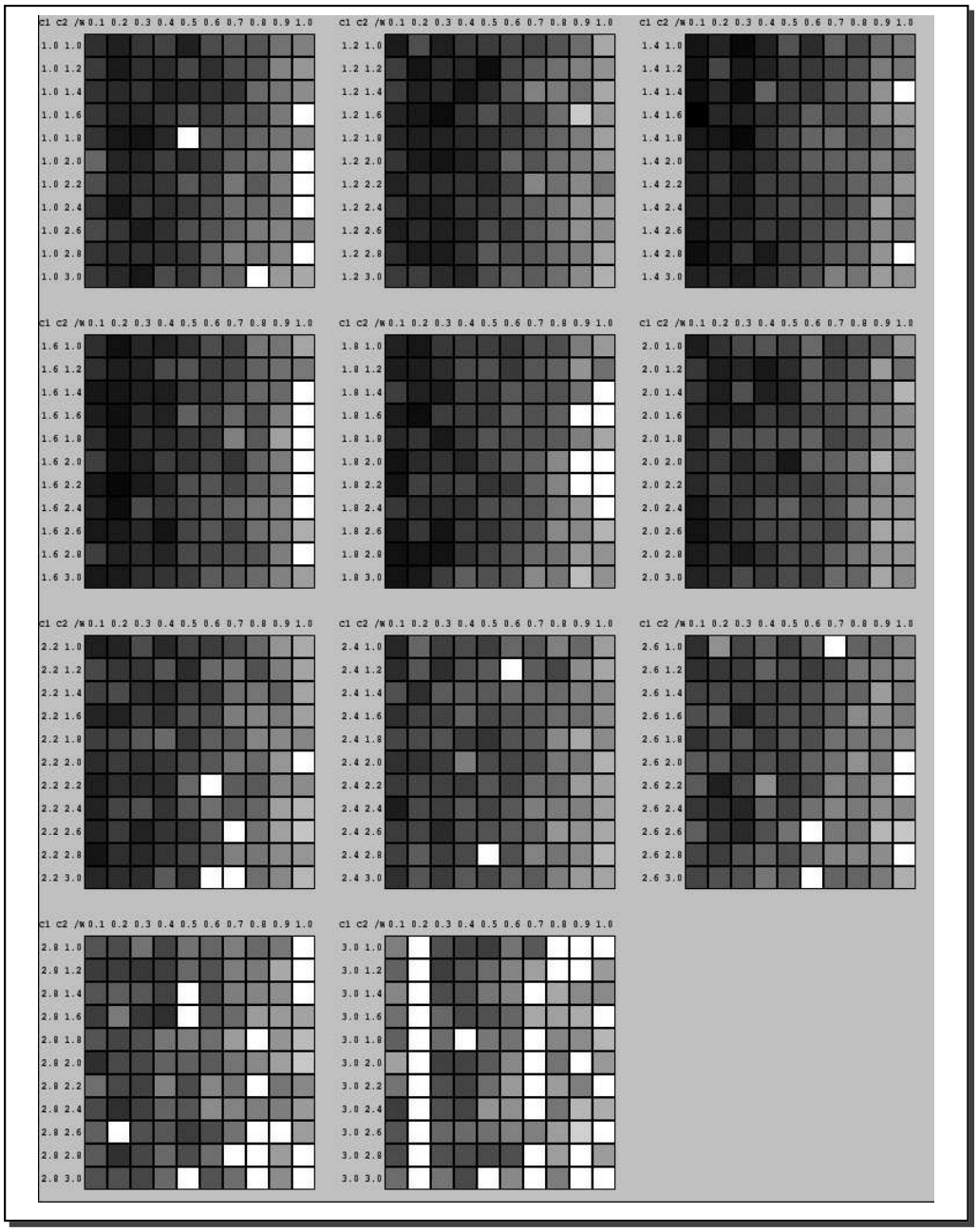

Figure B.28: Each square in the mosaic represents the average of inverted generational distance applied to 30 executions of Osyczka2's test function.

C

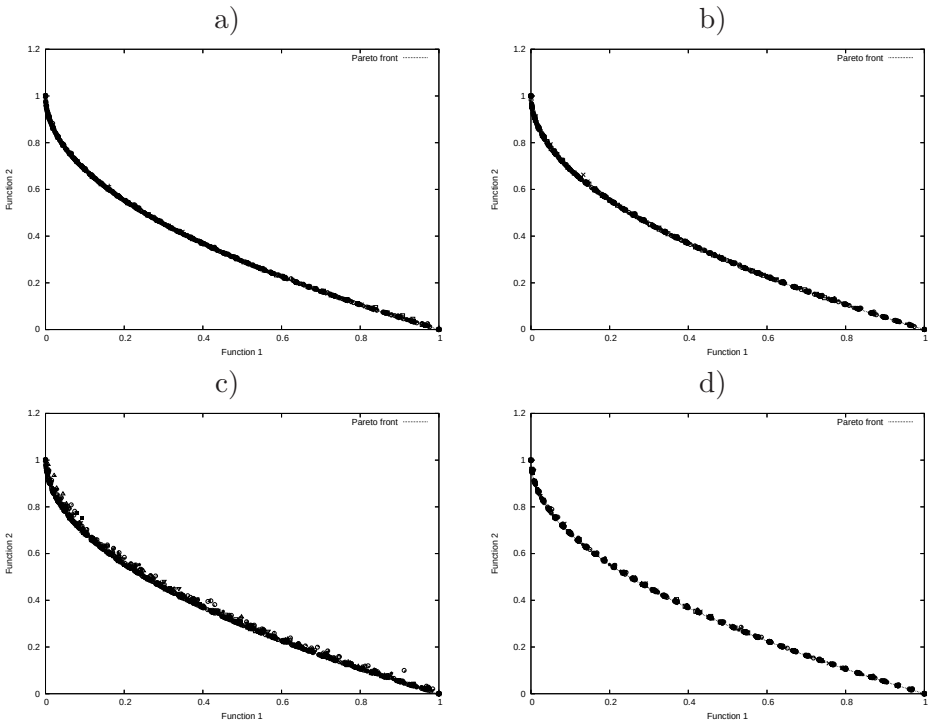

Figure C.1: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb1's test function.

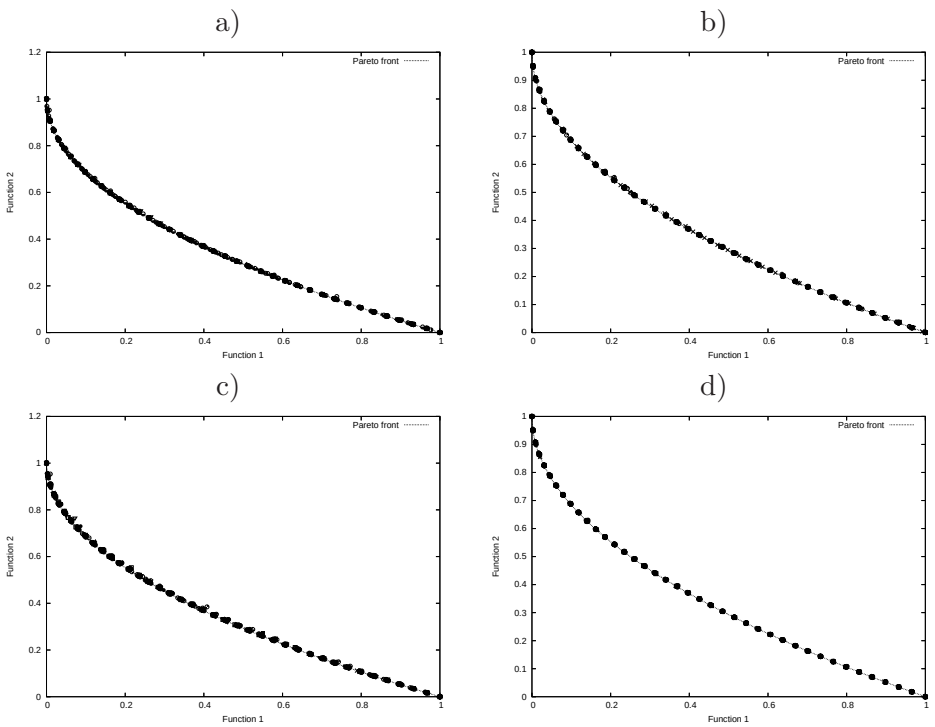

Figure C.2: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb1's test function.

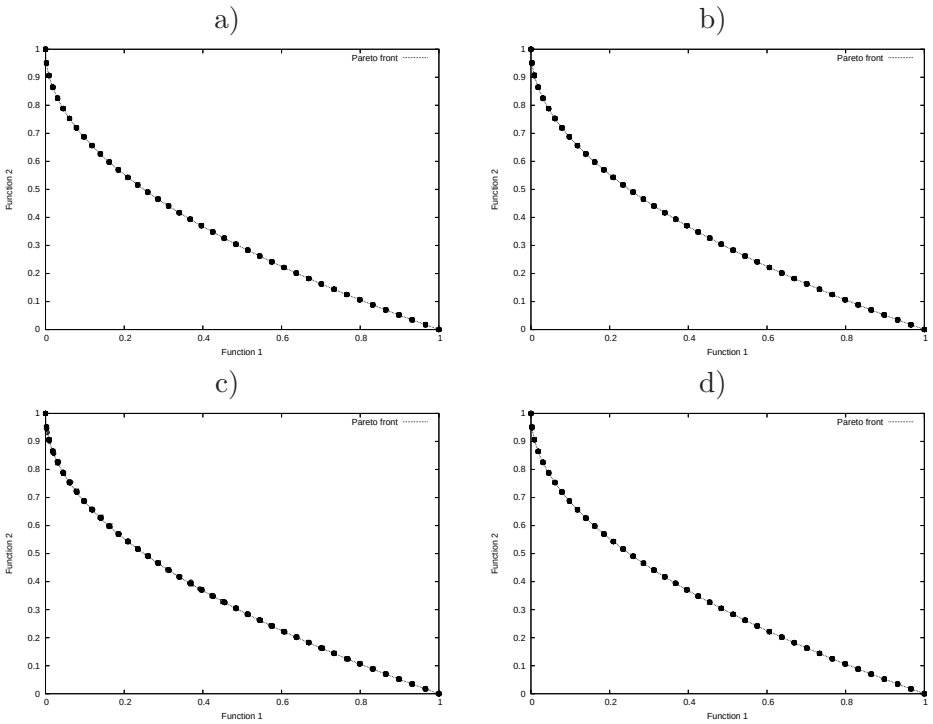

Figure C.3: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb1's test function.

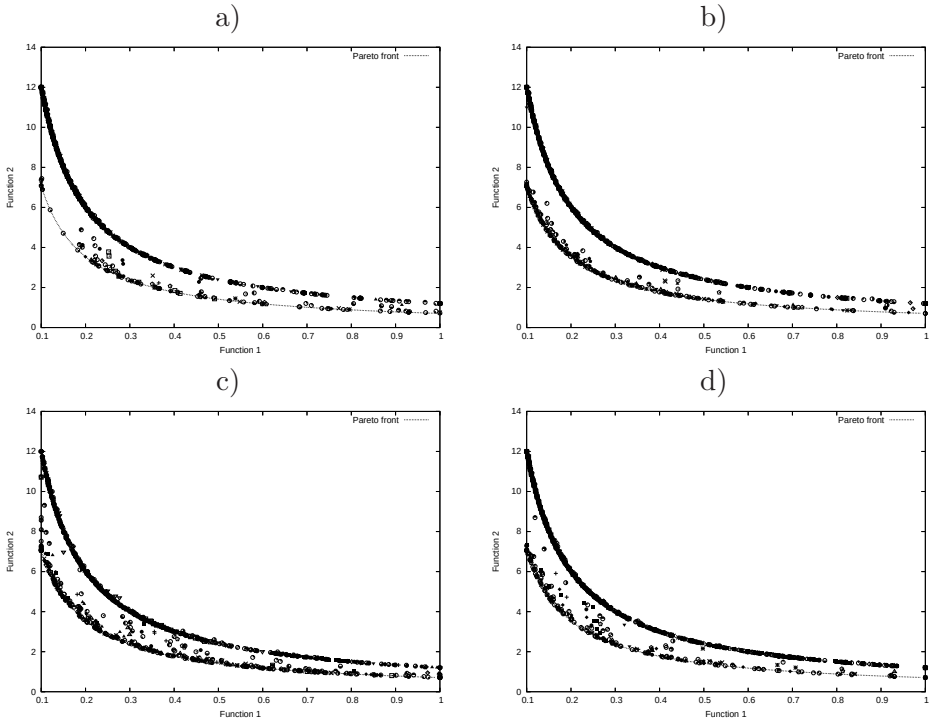

Figure C.4: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb2's test function.

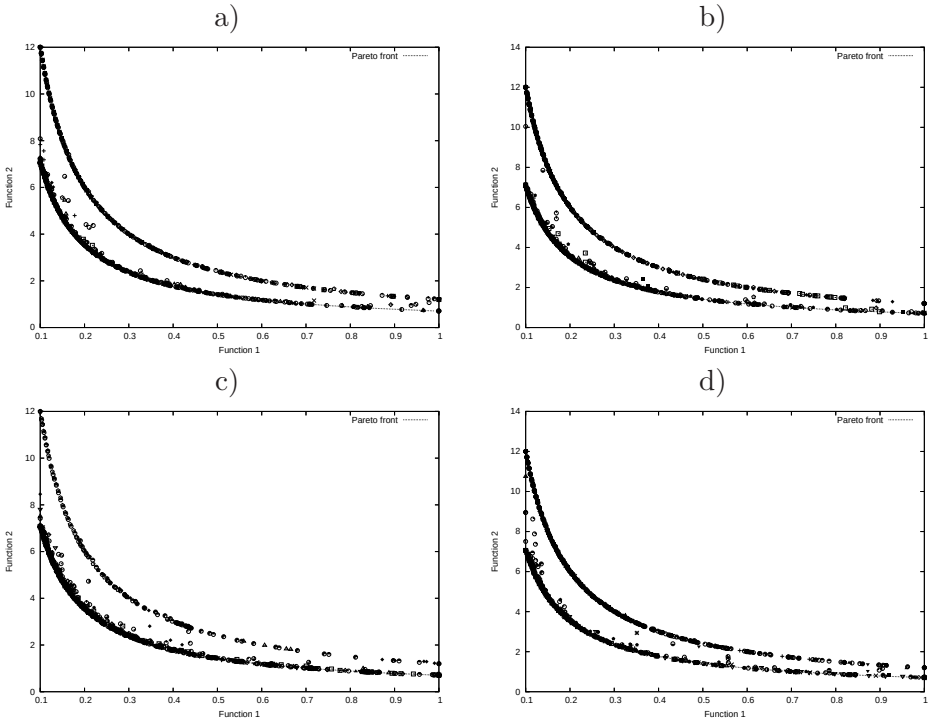

Figure C.5: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb2's test function.

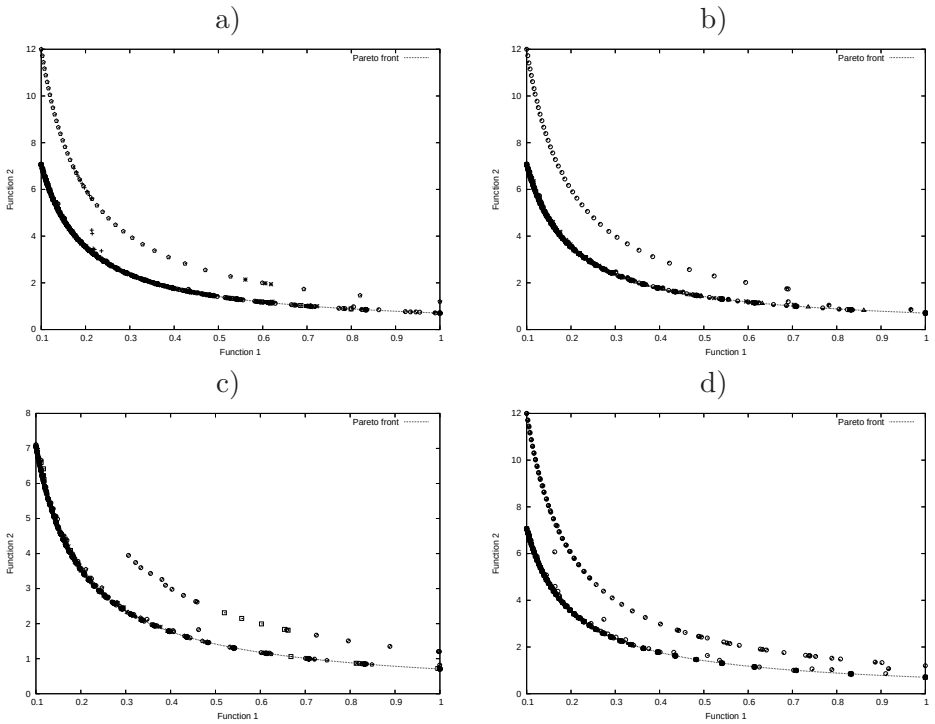

Figure C.6: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Deb2's test function.

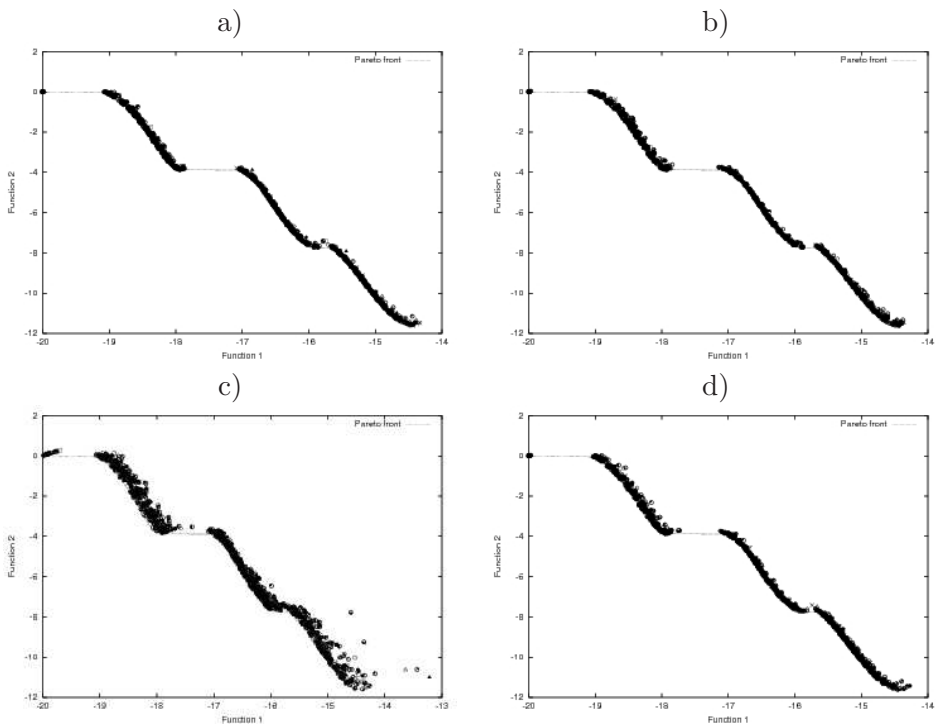

Figure C.7: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kursawe's test function.

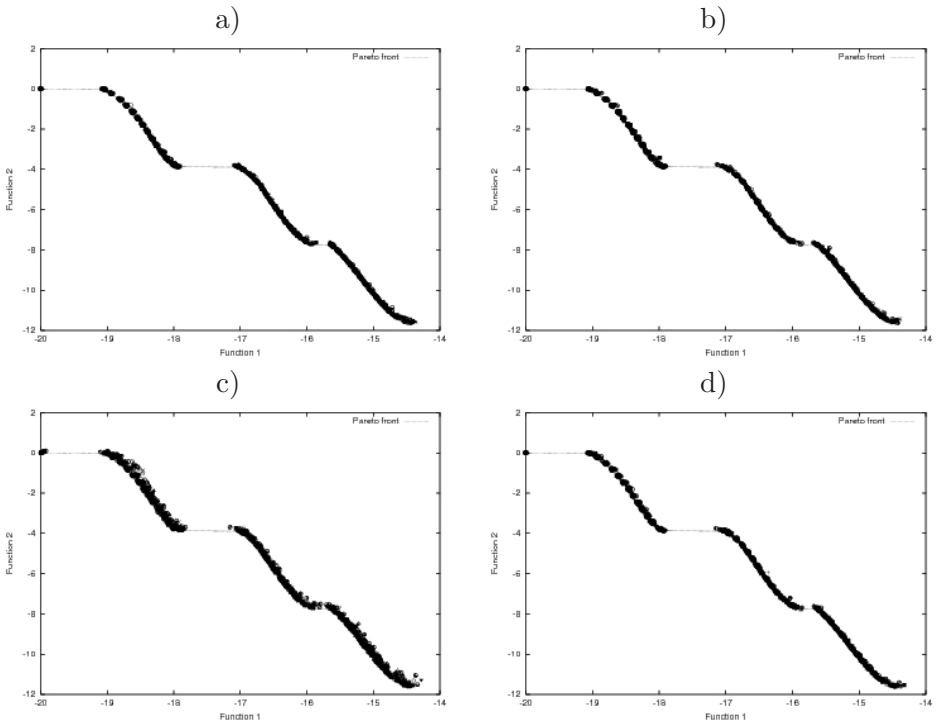

Figure C.8: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a fixed , b random, c self-adaptation and d self-adaptation (using half of the parameters' range) mechanisms for Kursawe's test function.

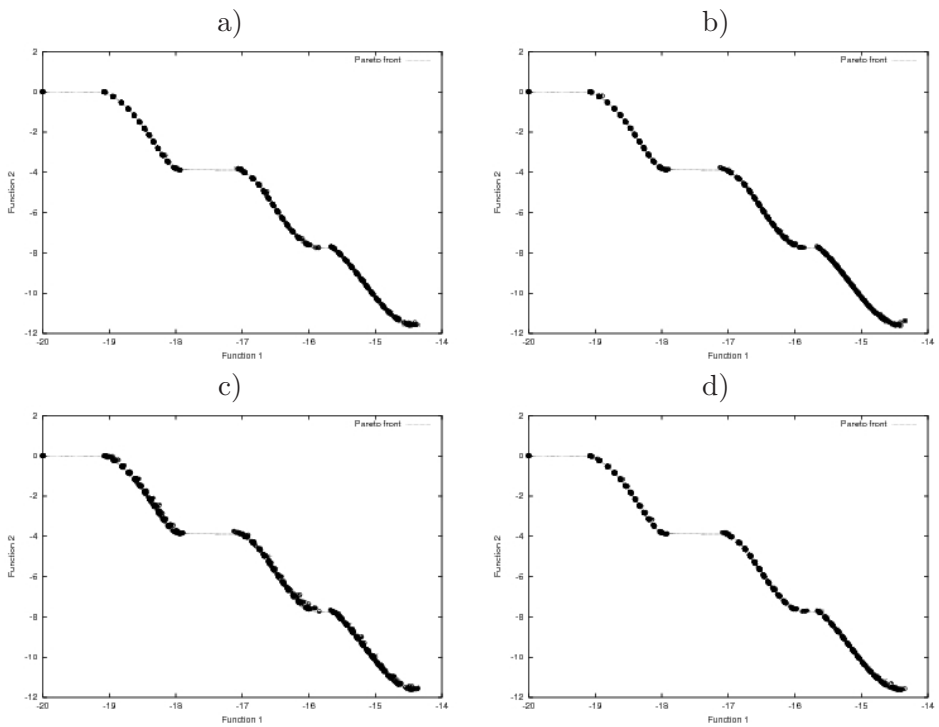

Figure C.9: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kursawe's test function.

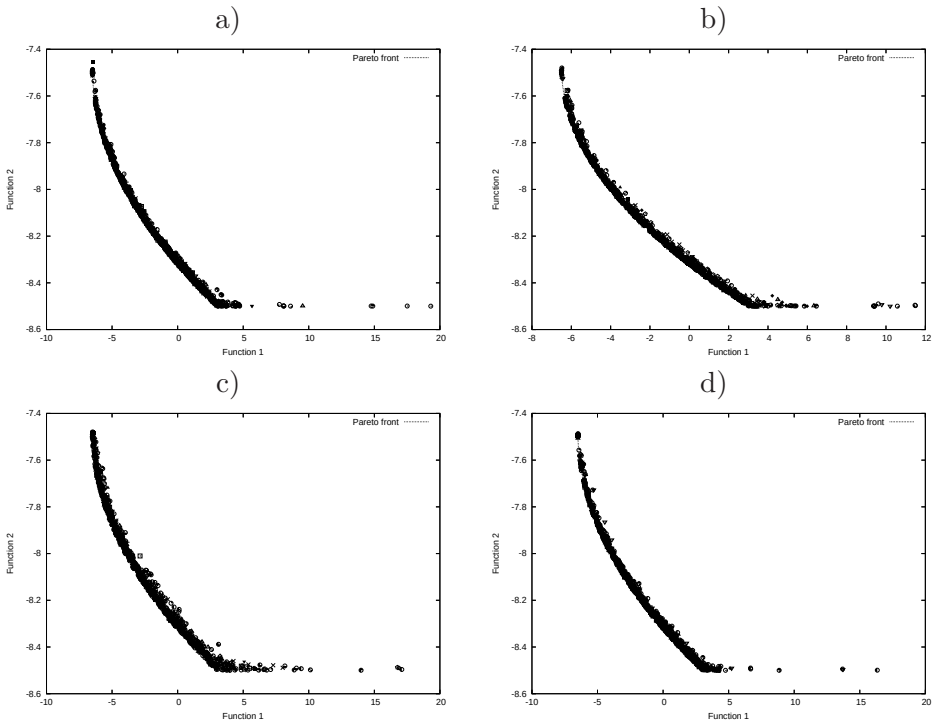

Figure C.10: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kita's test function.

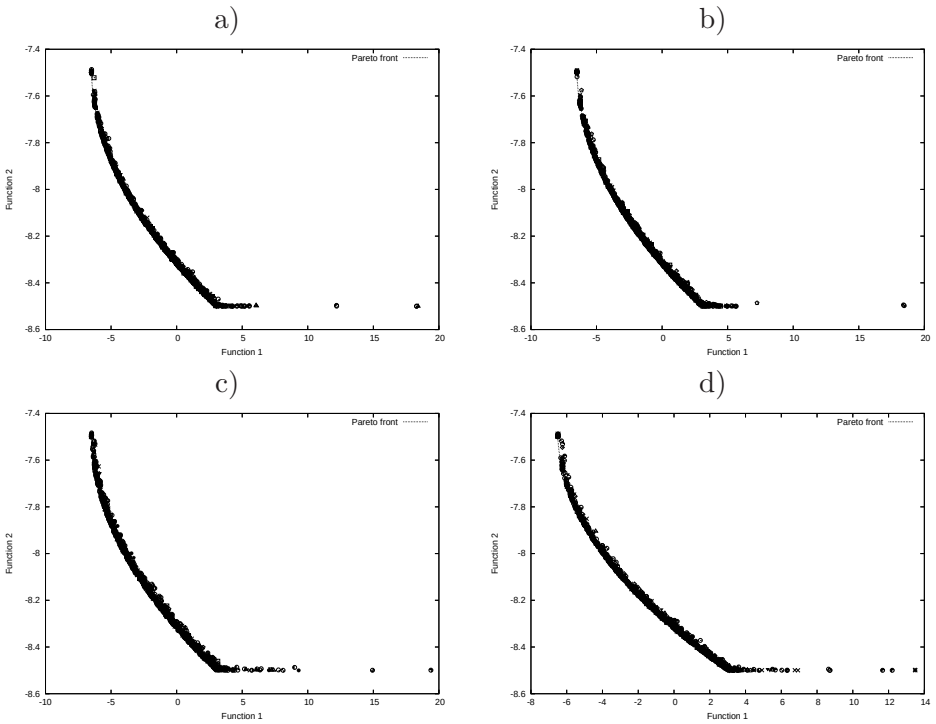

Figure C.11: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kita's test function.

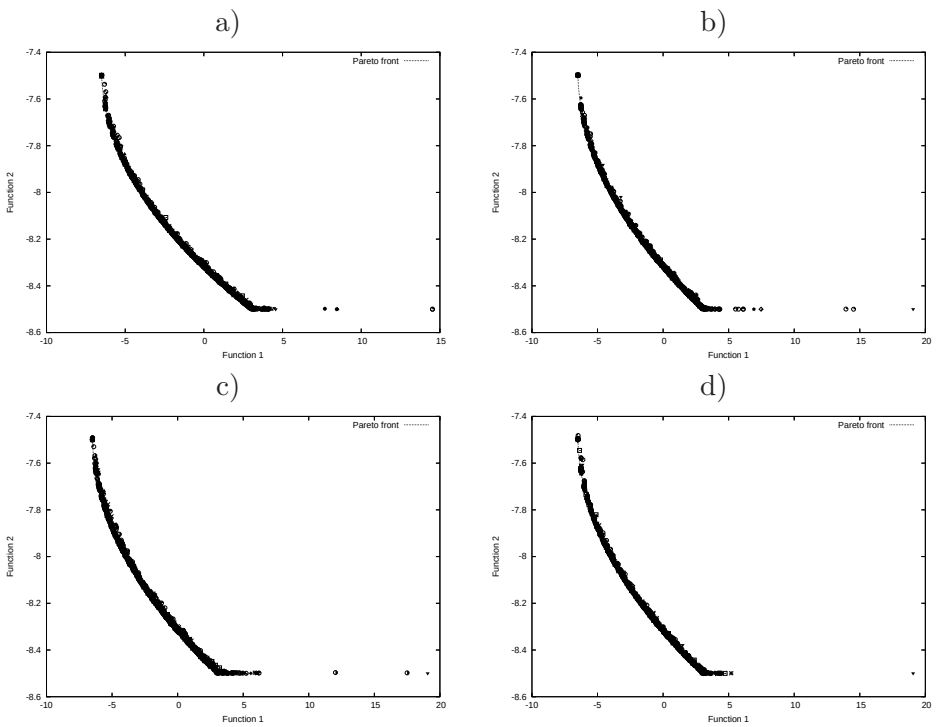

Figure C.12: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Kita's test function.

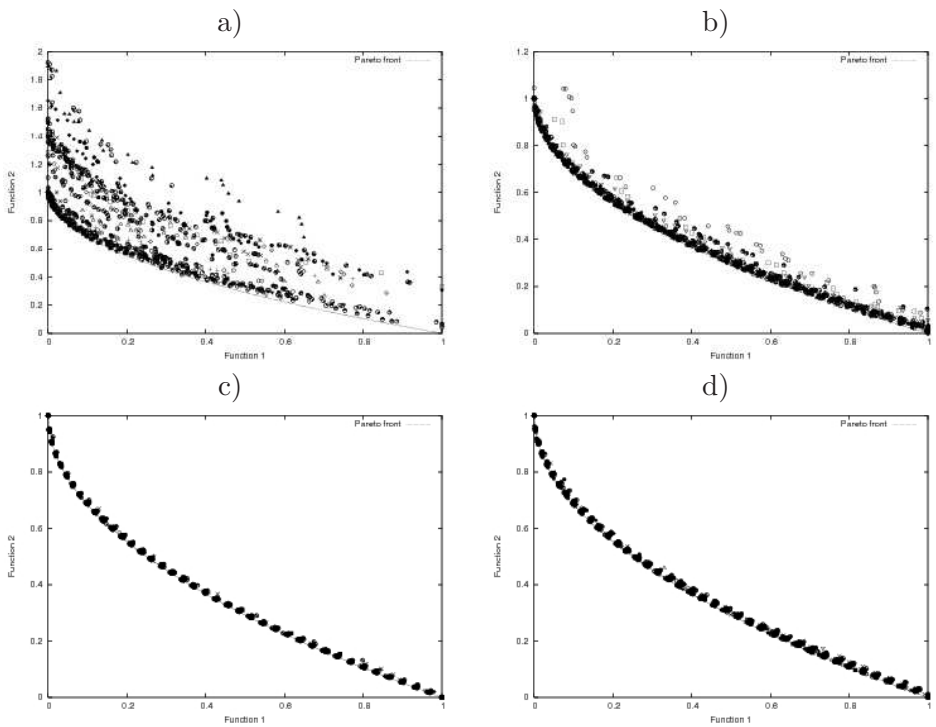

Figure C.13: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT1's test function.

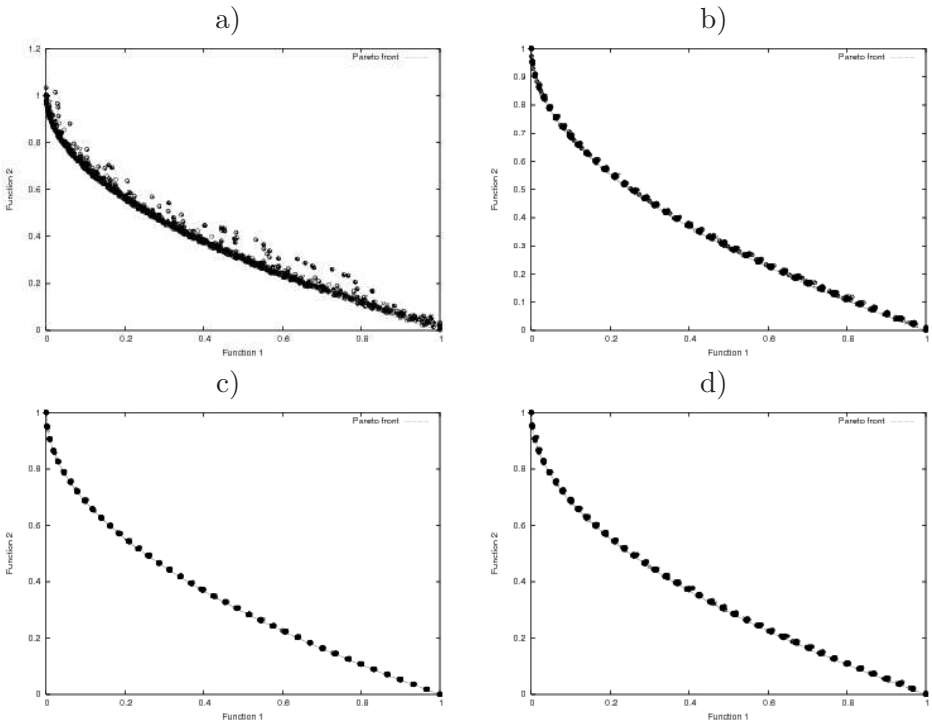

Figure C.14: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT1's test function.

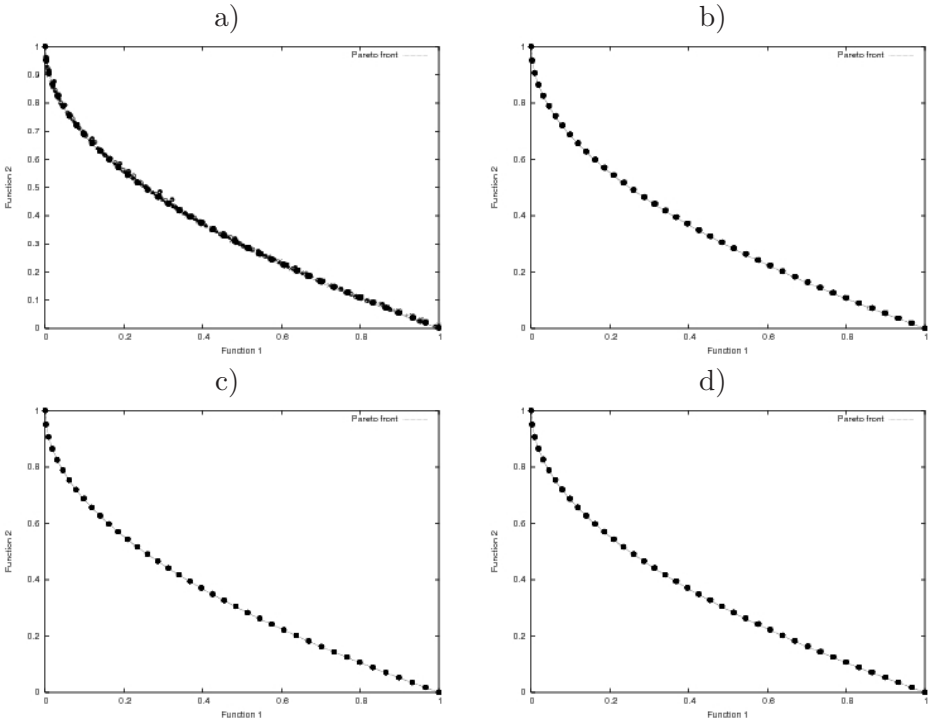

Figure C.15: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT1's test function.

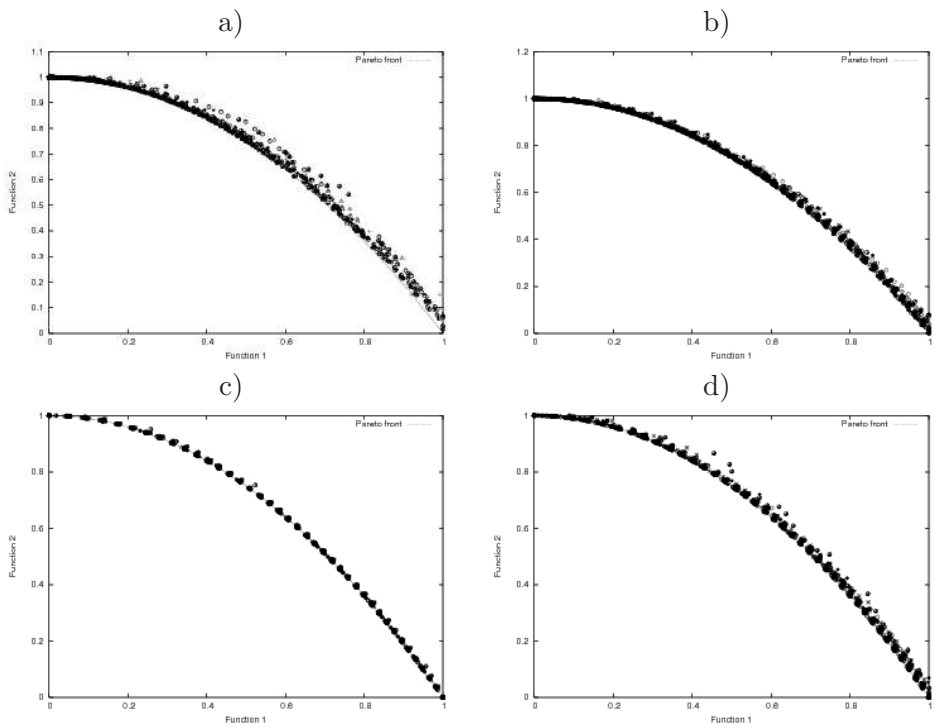

Figure C.16: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT2's test function.

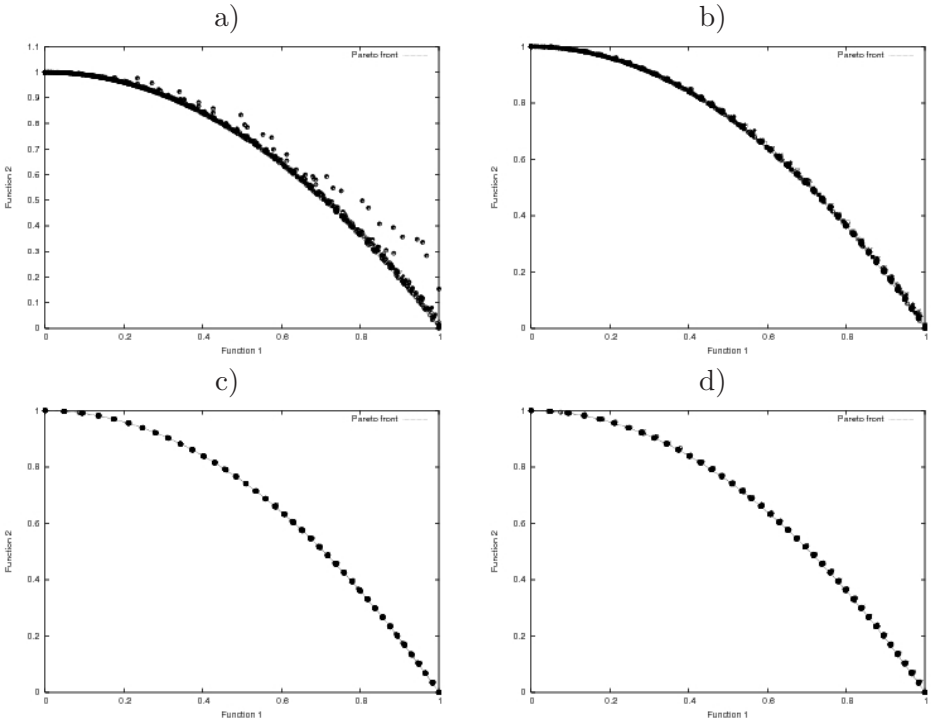

Figure C.17: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT2's test function.

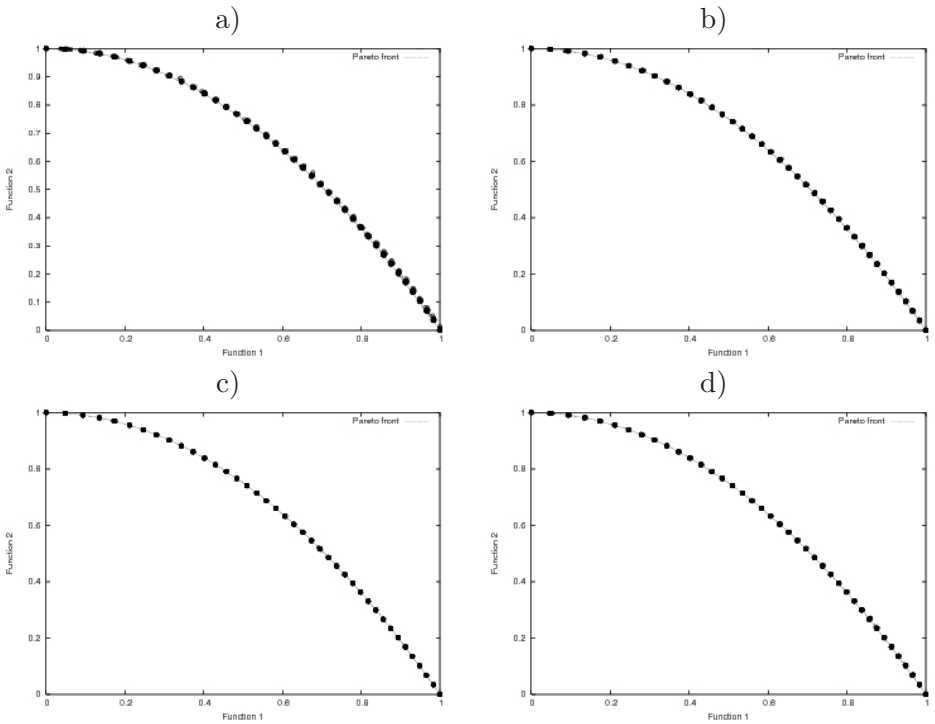

Figure C.18: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT2's test function.

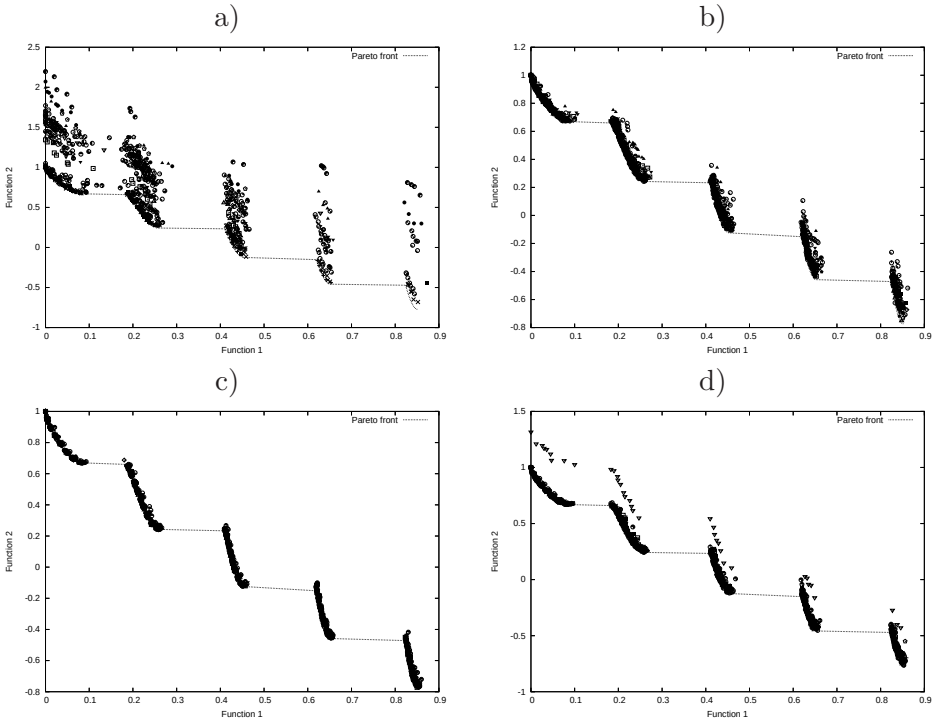

Figure C.19: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT3's test function.

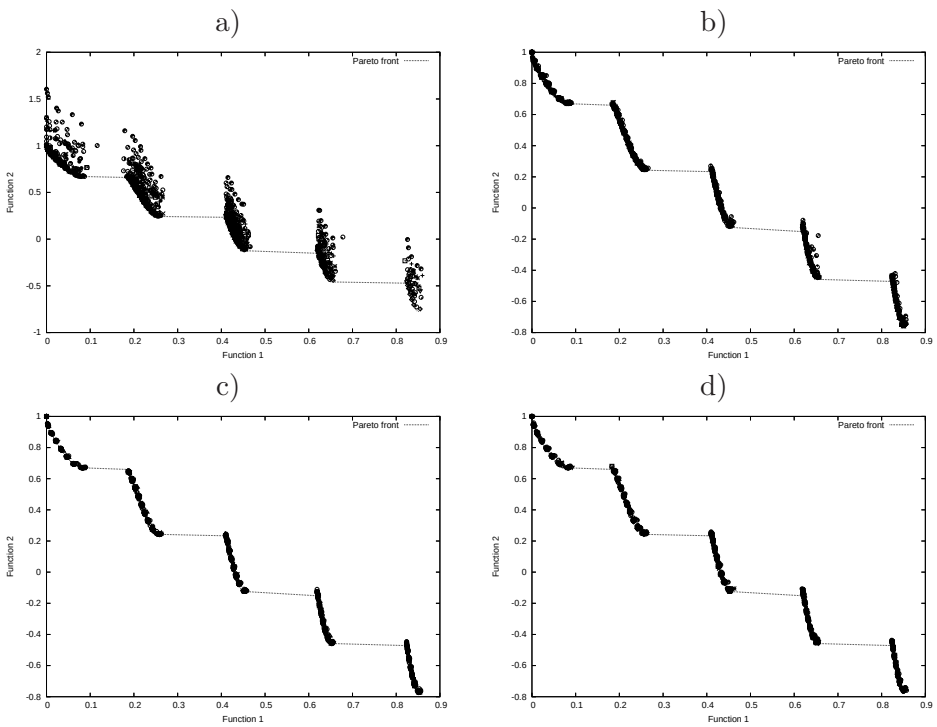

Figure C.20: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT3's test function.

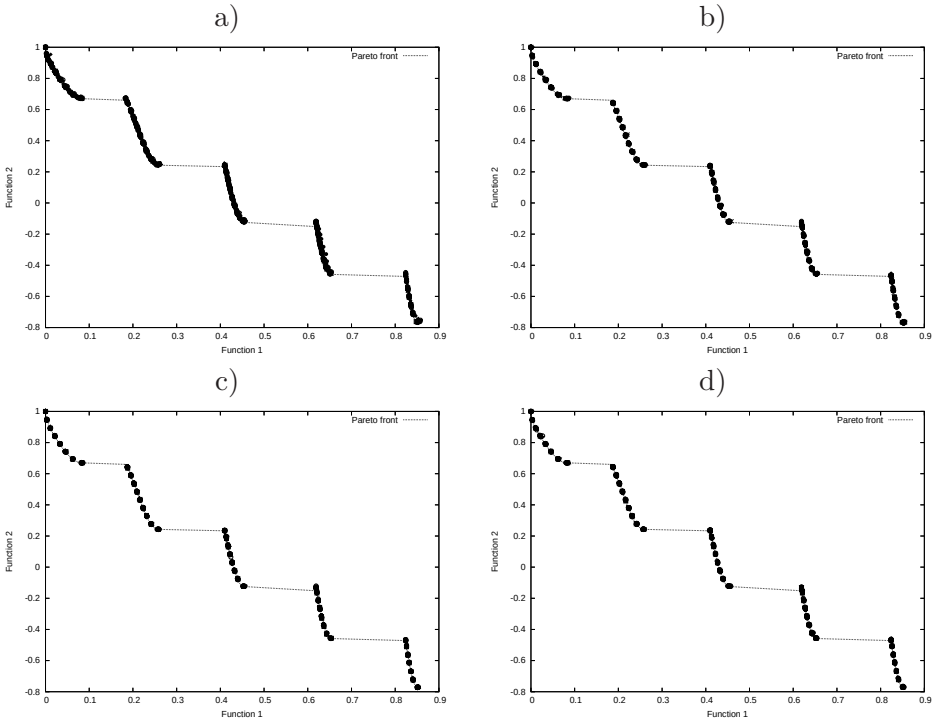

Figure C.21: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT3's test function.

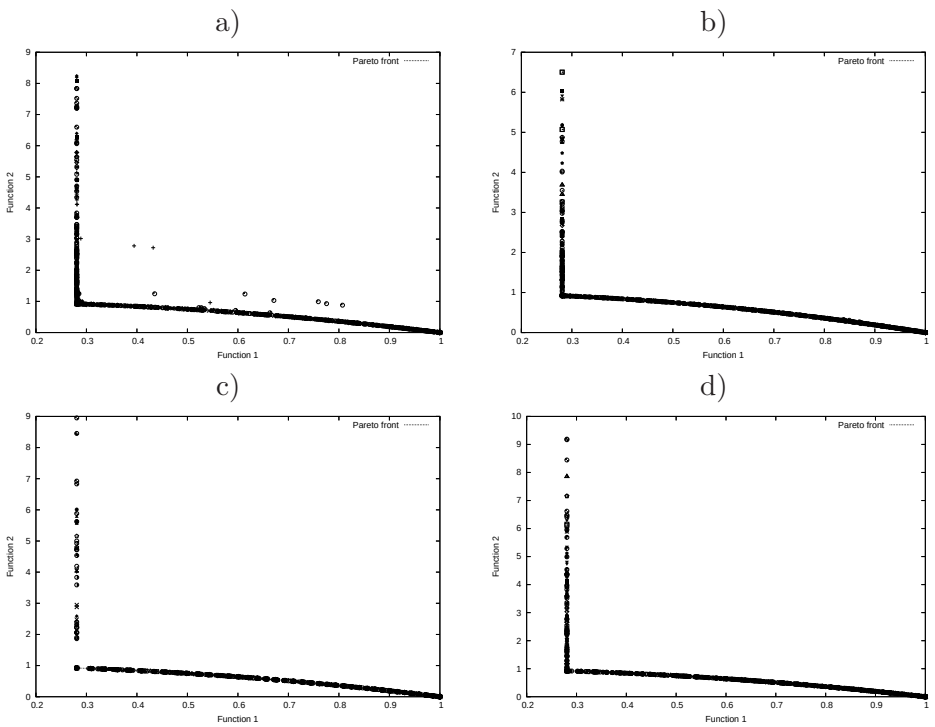

Figure C.22: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT6's test function.

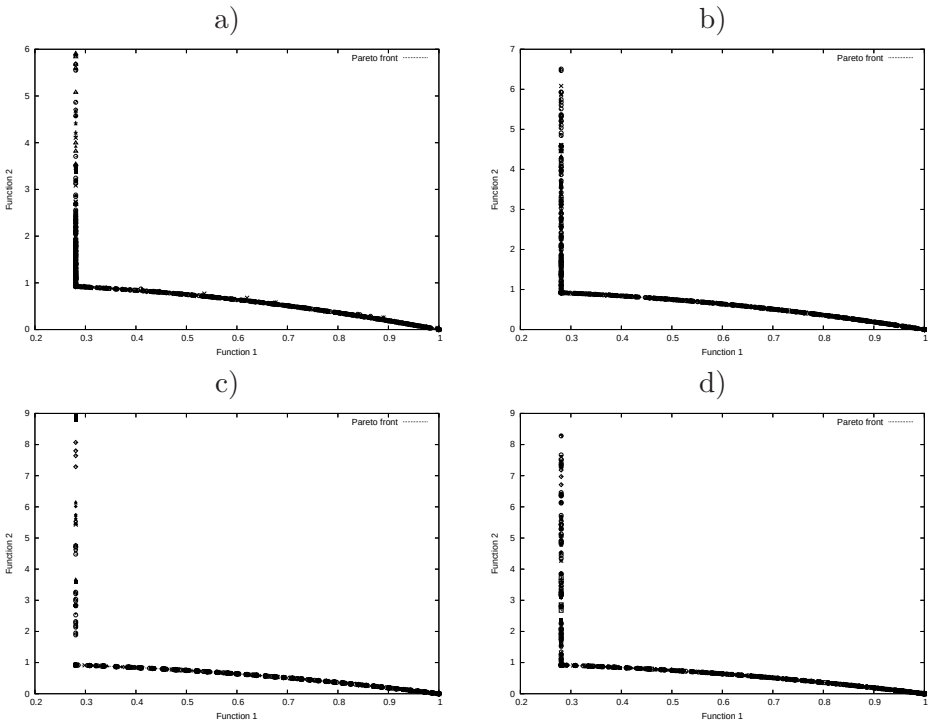

Figure C.23: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT6's test function.

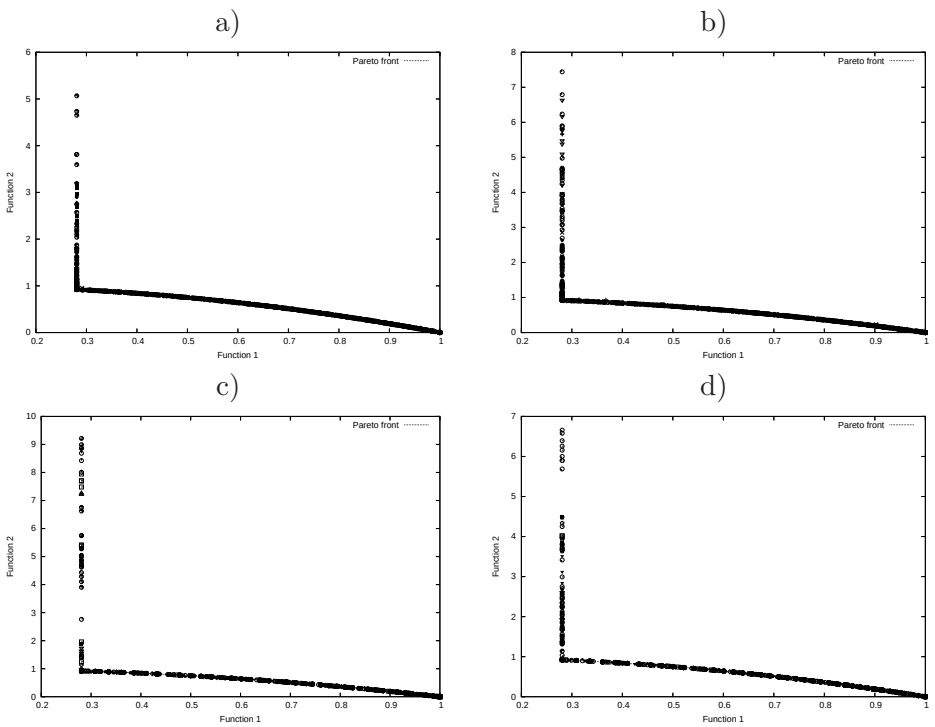

Figure C.24: Solutions produced using 250 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for ZDT6's test function.

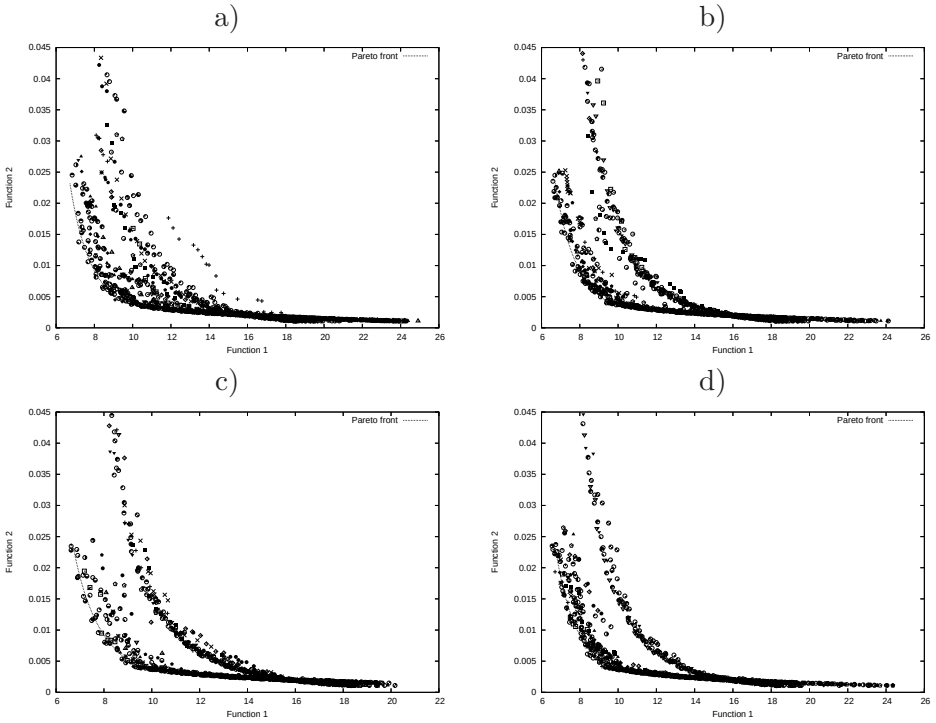

Figure C.25: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Welded Beam test function.

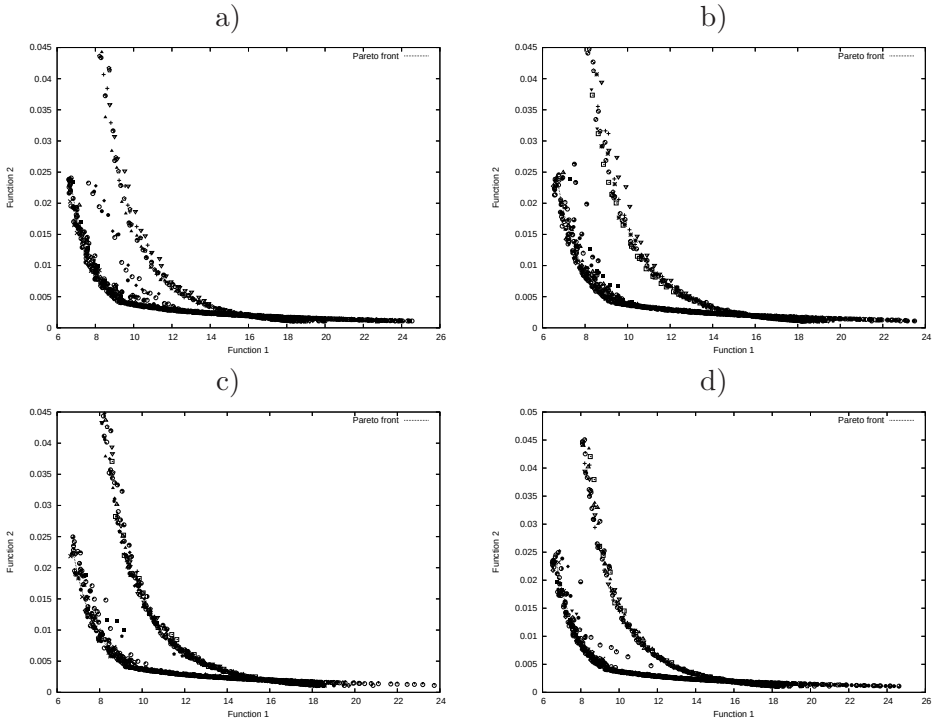

Figure C.26: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Welded Beam test function.

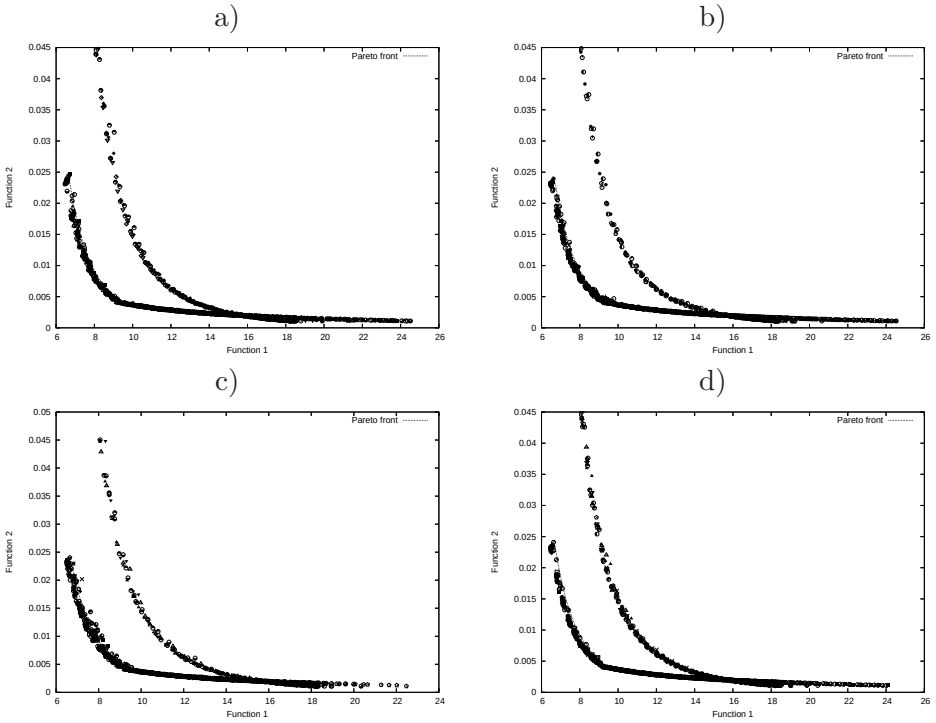

Figure C.27: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Welded Beam test function.

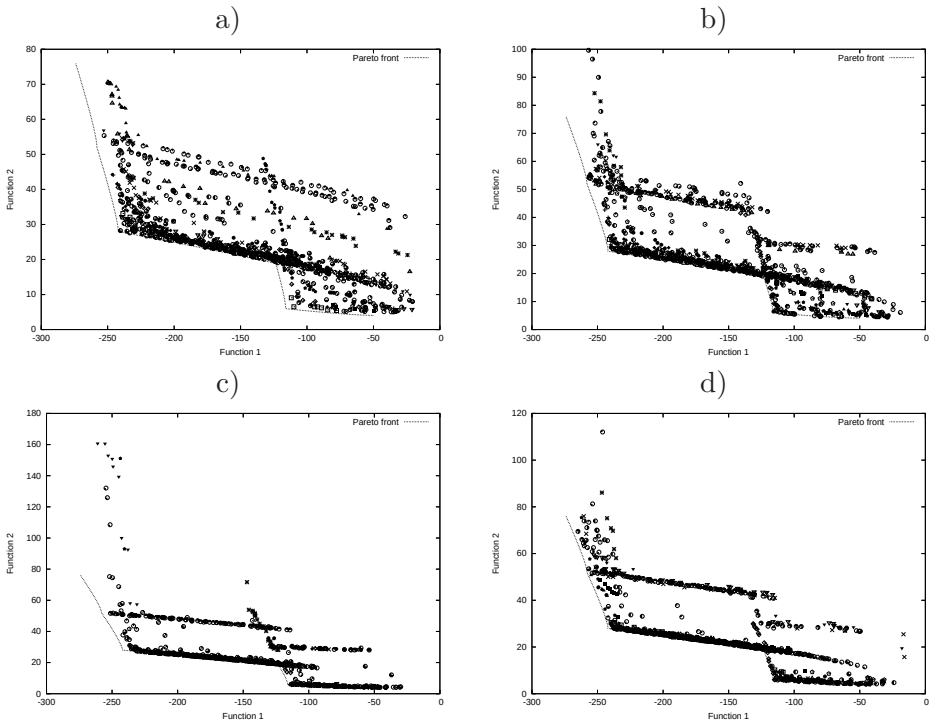

Figure C.28: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Osyczka 2's test function.

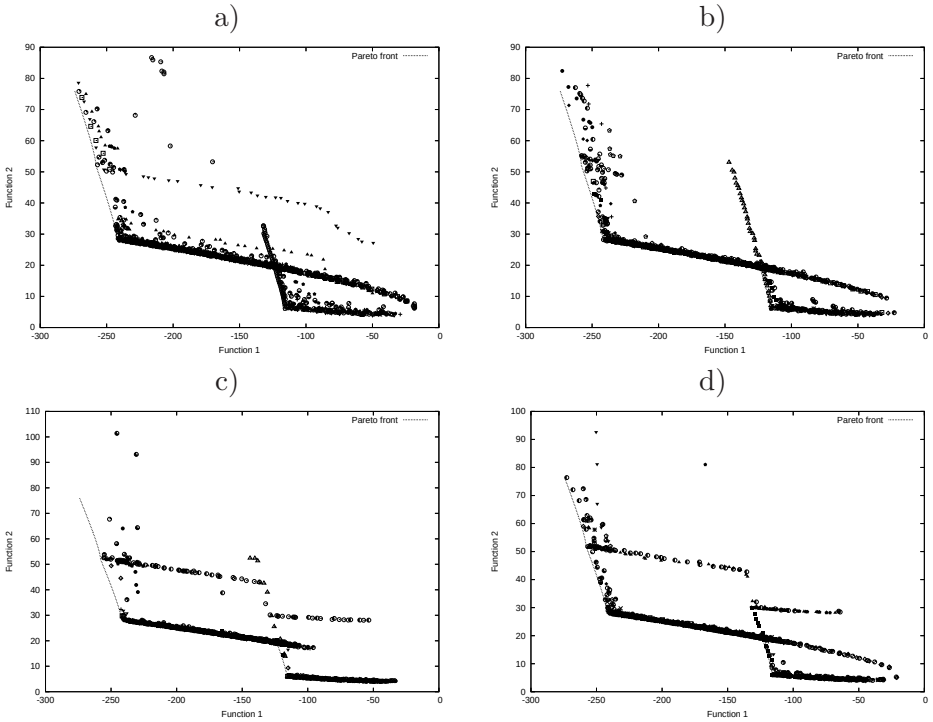

Figure C.29: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Osyczka 2's test function.

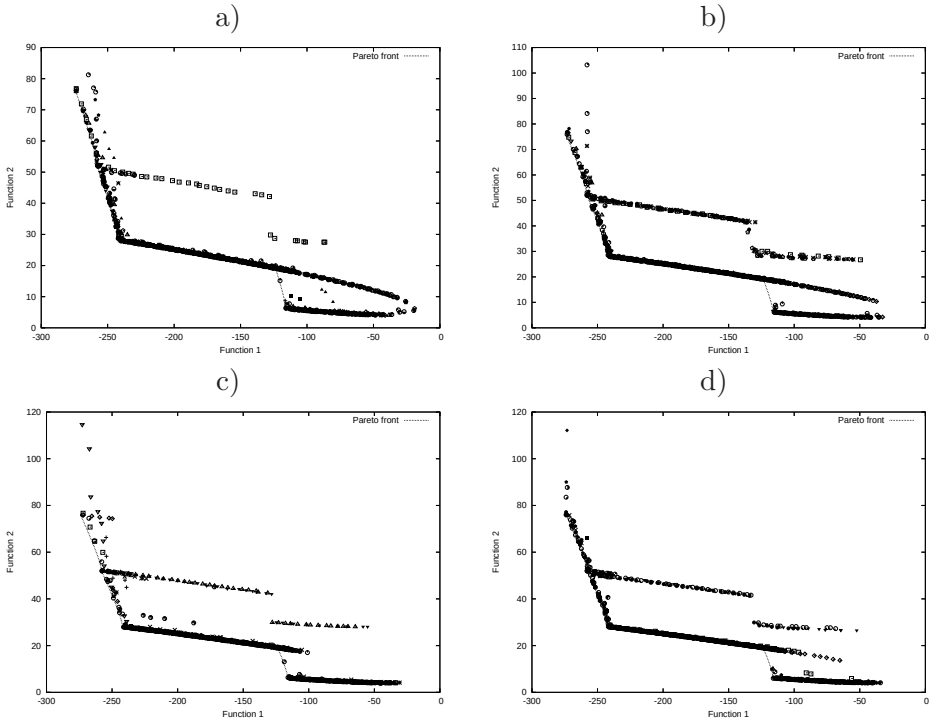

Figure C.30: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for Osyczka 2's test function.

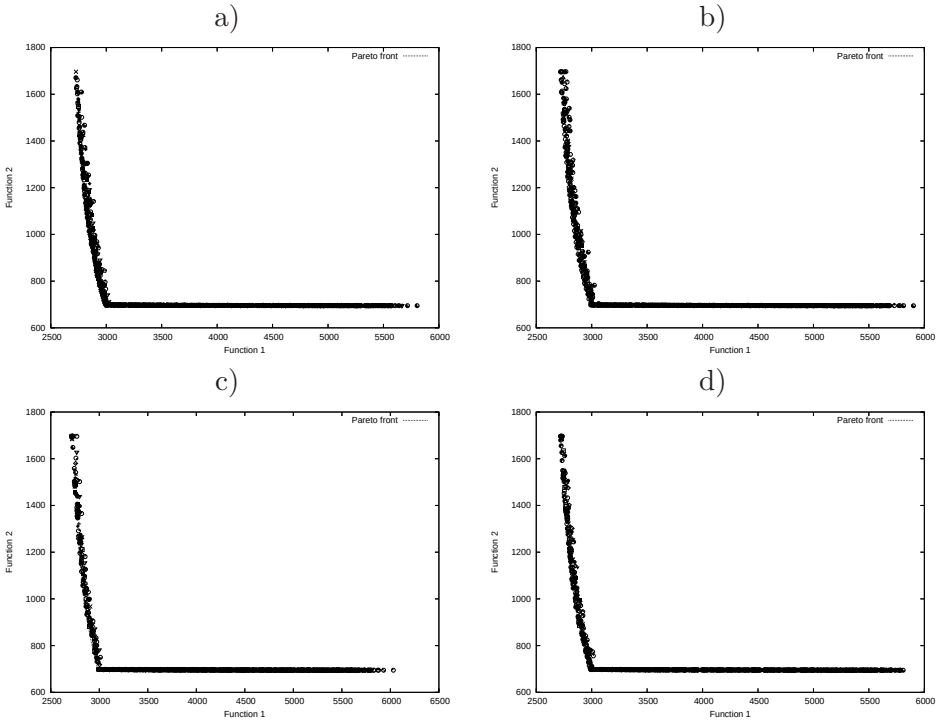

Figure C.31: Solutions produced using 50 iterations (2,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Speed Reducer test function.

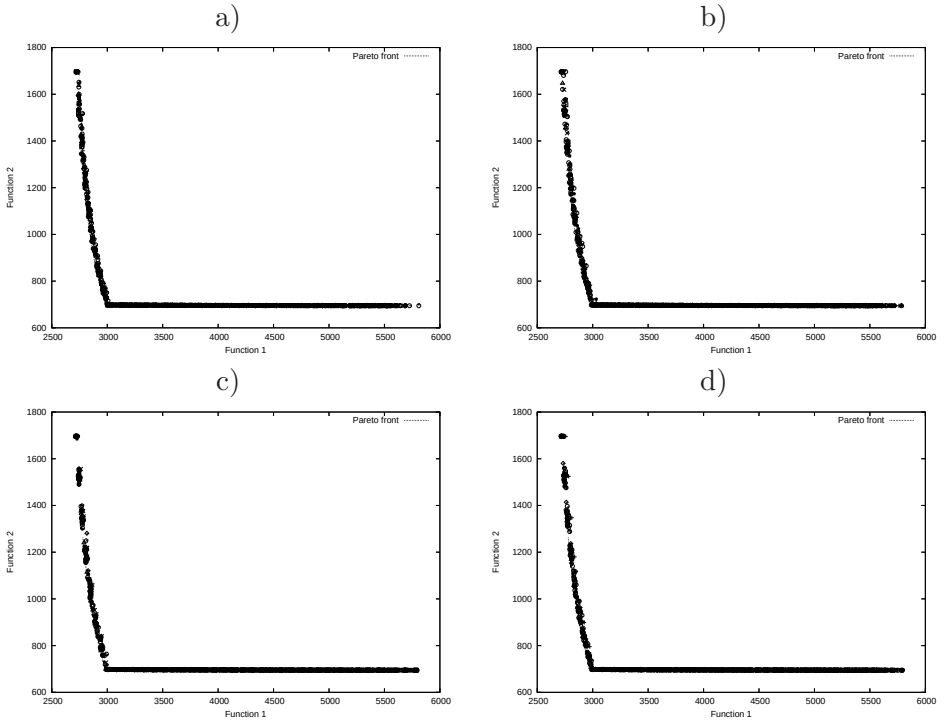

Figure C.32: Solutions produced using 100 iterations (4,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Speed Reducer test function.

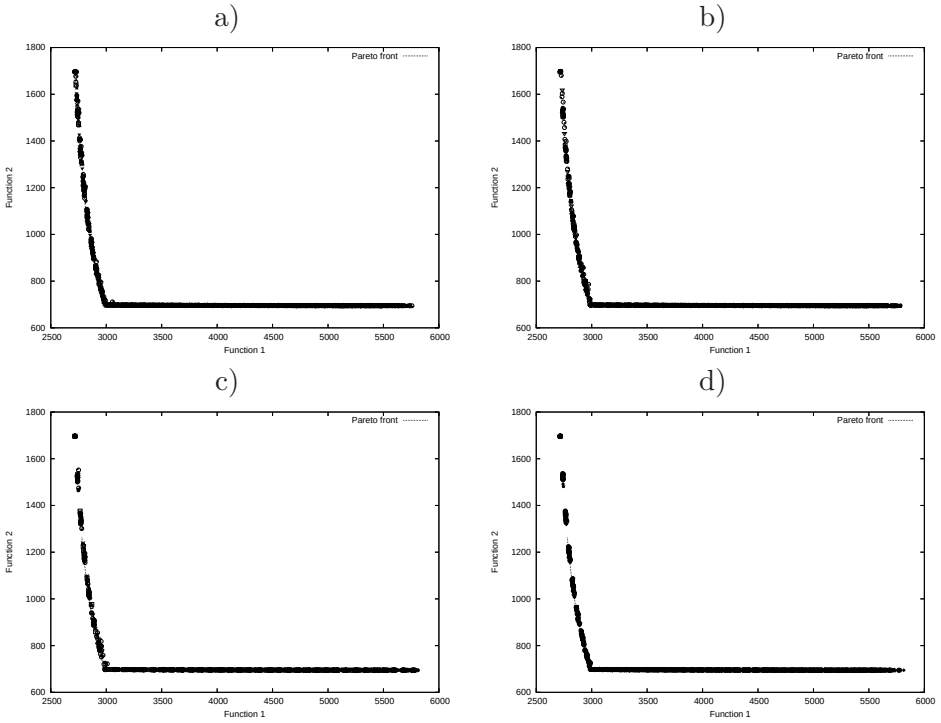

Figure C.33: Solutions produced using 250 iterations (10,000 fitness function evaluations) by our MOPSO using parameter selection by a) fixed , b) random, c) self-adaptation, d) self-adaptation (using half of the parameters' range) mechanisms for the Speed Reducer test function.

Bibliography

[1] Hussein A. Abbass. The Self-Adaptive Pareto Differential Evolution Algorithm. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 831–836, Piscataway, New Jersey, May 2002. IEEE Service Center.

[2] Johan Andersson. *Multiobjective Optimization in Engineering Design—Applications to Fluid Power Systems*. PhD thesis, Division of Fluid and Mechanical Engineering Systems. Department of Mechanical Engineering. Linköping Universitet, Linköping, Sweden, 2001.

[3] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, January 1998.

[4] Thomas Bartz-Beielstein, Philipp Limbourg, Konstantinos E. Parsopoulos, Michael N. Vrahatis, Jörn Mehnen, and Karlheinz Schmitt. Particle Swarm Optimizers for Pareto Optimization with Enhanced Archiving Techniques. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 3, pages 1780–1787, Canberra, Australia, December 2003. IEEE Press.

[5] U. Baumgartner, Ch. Magele, and W. Renhart. Pareto Optimality and Particle Swarm Optimization. *IEEE Transactions on Magnetics*, 40(2):1172–1175, March 2004.

[6] Peter Bienert. Aufbau einer optimierungsautomatik für drei parameter. Master's thesis, Universidad Técnica de Berlin, 1967.

[7] John T. Buchanan. A Naïve Approach for Solving MCDM Problems: The GUESS Method. *Journal of the Operational Research Society*, 48(2):202–206, 1997.

[8] Dirk Büche, Gianfranco Guidati, Peter Stoll, and Petros Kourmoursakos. Self-Organizing Maps for Pareto Optimization of Airfoils. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villaca nas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN VII*, pages 122–131, Granada, Spain, September 2002. Springer-Verlag. Lecture Notes in Computer Science No. 2439.

[9] Bill P. Buckles and Fred E. Petry. *Genetic Algorithms*. IEEE Computer Society Press, 1992.

[10] V. Chankong and Y.Y Haimes. The interactive surrogate worth trade-off (iswt) method for multiobjective decision-making. In S. Zionts, editor, *Multiple Criteria Problem Solving*, volume 155, pages 42–67, Berlin, Heidelberg, 1978. Springer–Verlag.

[11] A. Charnes, W.W. Cooper, and R.O. Ferguson. Optimal estimation of executive compensation by linear programming. *Management Science*, 1(No. 2):138–151, 1955.

[12] Genevieve Coath and Saman K. Halgamuge. A Comparison of Constraint-Handling Methods for the Application of Particle Swarm Optimization to Constrained Nonlinear Optimization Problems. In *Proceedings of the Congress on Evolutionary Computation 2003 (CEC'2003)*, volume 4, pages 2419–2425, Piscataway, New Jersey, December 2003. Canberra, Australia, IEEE Service Center.

[13] Carlos A. Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, 1(3):269–308, August 1999.

[14] Carlos A. Coello Coello. Evolutionary Multi-Objective Optimization: A Critical Review. In Ruhul Sarker, Masoud Mohammadian, and Xin Yao, editors, *Evolutionary Optimization*, pages 117–146. Kluwer Academic Publishers, New York, February 2002. ISBN 0-7923-7654-4.

[15] Carlos A. Coello Coello and Maximino Salazar Lechuga. MOPSO: A Proposal for Multiple Objective Particle Swarm Optimization. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1051–1056, Piscataway, New Jersey, May 2002. IEEE Service Center.

[16] Carlos A. Coello Coello and Gregorio Toscano Pulido. A Micro-Genetic Algorithm for Multiobjective Optimization. In Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 126–140. Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

[17] Carlos A. Coello Coello and Gregorio Toscano Pulido. Multiobjective Optimization using a Micro-Genetic Algorithm. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 274–282, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[18] Carlos A. Coello Coello, Gregorio Toscano Pulido, and Maximino Salazar Lechuga. Handling Multiple Objectives With Particle Swarm Optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, June 2004.

[19] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, May 2002. ISBN 0-3064-6762-3.

[20] N. L. Cramer. A representation for the adaptive generation of simple. In J.J. Grefenstette, editor, *Proceedings of an International Conference on Genetic*, pages 183–187, Pittsburgh, PA., 1985. Carnegie-Mellon University.

[21] Charles Darwin. *El origen de las especies*. Editorial Porrua, 1956.

[22] L. Davis. Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms, Arlington, VA.*, pages 61–69, 1989.

[23] Kalyanmoy Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation*, 7(3):205–230, Fall 1999.

[24] Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001. ISBN 0-471-87339-X.

[25] Kalyanmoy Deb and David E. Goldberg. An Investigation of Niche and Species Formation in Genetic Function Optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 42–50, San Mateo, California, June 1989. George Mason University, Morgan Kaufmann Publishers.

[26] Kalyanmoy Deb and Amarendra Kumar. Real-coded Genetic Algorithms with Simulated Binary Crossover: Studies on Multimodal and Multiobjective Problems. *Complex Systems*, 9:431–454, 1995.

[27] Kalyanmoy Deb, Manikanth Mohan, and Shikhar Mishra. Towards a Quick Computation of Well-Spread Pareto-Optimal Solutions. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 222–236, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[28] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[29] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable Test Problems for Evolutionary Multi-Objective Optimization. Technical Report 112, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 2001.

[30] A. K. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[31] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[32] Jonathan E. Fieldsend and Sameer Singh. A Multi-Objective Algorithm based upon Particle Swarm Optimisation, an Efficient Data Structure and Turbulence. In *Proceedings of the 2002 U.K. Workshop on Computational Intelligence*, pages 37–44, Birmingham, UK, September 2002.

[33] Peter C. Fishburn. Lexicographic orders, utilities and decision rules: A survey. *Management Science*, 20(No. 11):1442–1471, 1974.

[34] Terence C. Fogarty. Varying the probability of mutation in the genetic algorithm. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 104–109, San Mateo, CA, 1989. Morgan Kaufman.

[35] David B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. The Institute of Electrical and Electronic Engineers, New York, 1995.

[36] Lawrence J. Fogel. *On the organization of intellect*. PhD thesis, University of California, Los Angeles, California, 1964.

[37] Carlos M. Fonseca and Peter J. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 416–423, San Mateo, California, 1993. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

[38] Carlos M. Fonseca and Peter J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, 3(1):1–16, Spring 1995.

[39] Alex Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2002.

[40] S. Gass and T. L. Saaty. The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, 2:39–45, 1955.

[41] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms and Engineering Design*. John Wiley and Sons, Inc., New York, 1997.

[42] A. M. Geoffrion, J. S. Dyer, and A. Feinberg. An interactive approach for multi-criterion optimization, with an application to the operation of an academic department. *Management Science*, 19(4):357–368, 1972.

[43] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[44] David E. Goldberg and J. Richardson. Genetic algorithm with sharing for multimodal function optimization. In John J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Lawrence Erlbaum, 1987.

[45] John J. Grefenstette. GENESIS: A system for using genetic search procedures. In *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pages 161–165, 1984.

[46] John J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1):122–128, January/February 1986.

[47] Y. Y. Haimes, L. S. Lasdo, and D. A. Wismer. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man and Cybernetics*, 1:296–297, 1971.

[48] Sana Ben Hamida and Marc Schoenauer. ASCHEA: New Results Using Adaptive Segregational Constraint Handling. In *Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002)*, volume 1, pages 884–889, Piscataway, New Jersey, May 2002. IEEE Service Center.

[49] O. F. Herrera and M. Lozano. Adaptation of genetic algorithm parameters based on fuzzy logic controllers. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 95–125,. Physica-Verlag, 1996.

[50] J. Hesser and R. Manner. Towards an optimal mutation probability for genetic algorithms. *Parallel Problem Solving from Nature*, 496:23–32, 1990.

[51] R. Hinterding, Z. Michalewicz, and A.E. Eiben. Adaptation in evolutionary computation: a survey. In *Proceedings of the Fourth IEEE Conference on Evolutionary Computation, Indianapolis, IN*, pages 65–69, 1997.

[52] John H. Holland. *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Ann Harbor : University of Michigan Press, 1975.

[53] John H. Holland. *Adaptation in Natural an Artificial Systems*. MIT Press, Cambridge, Massachusetts, second edition, 1992.

[54] Jeffrey Horn and Nicholas Nafpliotis. Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.

[55] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 82–87, Piscataway, New Jersey, June 1994. IEEE Service Center.

[56] Xiaohui Hu and Russell Eberhart. Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pages 1677–1681, Piscataway, New Jersey, May 2002. IEEE Service Center.

[57] Xiaohui Hui, Russell C. Eberhart, and Yuhui Shi. Particle Swarm with Extended Memory for Multiobjective Optimization. In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pages 193–197, Indianapolis, Indiana, USA, April 2003. IEEE Service Center.

[58] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[59] A. Jaszkiewicz and R. Slowinski. The light beam search over a non-dominated surface of a multiple-objective programming problem. In G.H. Tzeng, H.F. Wand, U.P. Wend, and P.L. Yu, editors, *Proceedings of the Tenth International Conference: Expand and Enrich the Domains of Thinking and Application*, pages 87–99, New York, 1994. Springer–Verlag.

[60] Andrzej Jaszkiewicz. On the Performance of Multiple-Objective Genetic Local Search on the 0/1 Knapsack Problem—A Comparative Experiment. *IEEE Transactions on Evolutionary Computation*, 6(4):402–412, August 2002.

[61] Mikkel T. Jensen. Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms. *IEEE Transactions on Evolutionary Computation*, 7(5):503–515, October 2003.

[62] Yaochu Jin, Tatsuya Okabe, and Bernhard Sendhoff. Dynamic Weighted Aggregation for Evolutionary Multi-Objective Optimization: Why Does It Work and How? In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 1042–1049, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[63] S.C. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, 32:241–254, 1967.

[64] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. John Wiley and Sons, New York, 1976.

[65] James Kennedy and Russell C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, California, 2001.

[66] H. Kestelman. *Modern Theories of Integration*, chapter Chapter 3 Lebesgue Measure, pages 67–91. New York:Dover, 2nd edition, 1960.

[67] Chi kin Chow and Hung tat Tsui. Autonomous Agent Response Learning by a Multi-Species Particle Swarm Optimization. In *2004 Congress on Evolutionary Computation (CEC'2004)*, volume 1, pages 778–785, Portland, Oregon, USA, June 2004. IEEE Service Center.

[68] Hajime Kita, Yasuyuki Yabumoto, Naoki Mori, and Yoshikazu Nishikawa. Multi-Objective Optimization by Means of the Thermodynamical Genetic Algorithm. In

Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature—PPSN IV*, Lecture Notes in Computer Science, pages 504–512, Berlin, Germany, September 1996. Springer-Verlag.

[69] Joshua Knowles and David Corne. Properties of an Adaptive Archiving Algorithm for Storing Nondominated Vectors. *IEEE Transactions on Evolutionary Computation*, 7(2):100–116, April 2003.

[70] Joshua D. Knowles. *Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization*. PhD thesis, The University of Reading, Department of Computer Science, Reading, UK, January 2002.

[71] Joshua D. Knowles and David W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 8(2):149–172, 2000.

[72] Teuvo Kohonen, T.S. Huang, and M.R. Schroeder, editors. *Self-Organizing Maps*. Springer-Verlag, 2001.

[73] P. Korhonen and J. Laakso. A visual interactive method for solving the multiple criteria problem. In *Lecture Notes in Economics and Mathematical Systems*, volume 229, pages 145–153. M. Grauer and A.P. Wierzbicki, Springer–Verlag, 1984.

[74] John R. Koza. Herarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan, editor, *Proceedings of 11th International Joint Conference on Artificial Intelligence*, pages 768–774, San Mateo, California, 1989. Morgan Kaufmann.

[75] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1992.

[76] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[77] Frank Kursawe. A Variant of Evolution Strategies for Vector Optimization. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature. 1st Workshop, PPSN I*, volume 496 of *Lecture Notes in Computer Science Vol. 496*, pages 193–197, Berlin, Germany, October 1991. Springer-Verlag.

[78] Marco Laumanns, Günter Rudolph, and Hans-Paul Schwefel. Mutation Control and Convergence in Evolutionary Multi-Objective Optimization. In *Proceedings of the 7th International Mendel Conference on Soft Computing (MENDEL 2001)*, Brno, Czech Republic, June 2001.

[79] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining Convergence and Diversity in Evolutionary Multi-objective Optimization. *Evolutionary Computation*, 10(3):263–282, Fall 2002.

[80] Marco Laumanns, Lothar Thiele, Eckart Zitzler, and Kalyanmoy Deb. Archiving with Guaranteed Convergence and Diversity in Multi-Objective Optimization. In W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 439–447, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

[81] Xiaodong Li. A Non-dominated Sorting Particle Swarm Optimizer for Multiobjective Optimization. In Erick Cantú-Paz et al., editor, *Genetic and Evolutionary Computation—GECCO 2003. Proceedings, Part I*, pages 37–48. Springer. Lecture Notes in Computer Science Vol. 2723, July 2003.

[82] Mahdi Mahfouf, Min-You Chen, and Derek Arturh Linkens. Adaptive Weighted Particle Swarm Optimisation for Multi-objective Optimal Design of Alloy Steels. In *Parallel Problem Solving from Nature - PPSN VIII*, pages 762–771, Birmingham, UK, September 2004. Springer-Verlag. Lecture Notes in Computer Science Vol. 3242.

[83] M.M. Mäkelä. Issues of implementing a fortran subroutine package nsolib for nonsmooth optimization. Technical Report Report 5/1993, University of Jyväskylä, Department of Mathematics, Laboratory of Scientific Computing, Jyväskylä, 1993.

[84] Gregor Johann Mendel. Experiments in plant hybridisation. *Journal of Royal Horticultural Society*, 26:1–32, 1901. Traducción al inglés de un artículo publicado.

[85] Efrén Mezura-Montes and Carlos A. Coello Coello. A Simple Multimembered Evolution Strategy to Solve Constrained Optimization Problems. Technical Report EVOCINV-04-2003, Evolutionary Computation Group at CINVESTAV, Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México D.F., México, 2003. Available in the Constraint Handling Techniques in Evolutionary Algorithms Repository at http://www.cs.cinvestav.mx/˜constraint/.

[86] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[87] Kaisa M. Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston, Massachusetts, 1998.

[88] Kaisa M. Miettinen and M.M. Mäkelä. A nondifferentiable multiple criteria optimization method applied to continuous casting process. In A. Fasano, M primicerio, and B.B. Teubner, editors, *Proceedings of the Seventh European Conference on Mathematics in Industry*, pages 255–262. Stuttgart, 1994.

[89] Jacqueline Moore and Richard Chapman. Application of Particle Swarm to Multiobjective Optimization. Department of Computer Science and Software Engineering, Auburn University. (Unpublished manuscript), 1999.

[90] J.N. Morse. Reducing the size of the nondominated set: Pruning by clustering. *Computers and Operations Research*, 7(1–2):55–66, 1980.

[91] Sanaz Mostaghim and Jürgen Teich. Strategies for Finding Good Local Guides in Multi-objective Particle Swarm Optimization (MOPSO). In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pages 26–33, Indianapolis, Indiana, USA, April 2003. IEEE Service Center.

[92] Andrzej Osyczka. Multicriteria optimization for engineering design. In John S. Gero, editor, *Design Optimization*, pages 193–227. Academic Press, 1985.

[93] Andrzej Osyczka and Sourav Kundu. Using genetic algorithms to solve multicriteria nonlinear programming problems. In *Proceedings of Mendel'95, the 1st International Mendel Conference on Genetic Algorithms*, Brno, Czech Republic, September 1995.

[94] Luis Paquete and Thomas Stützle. A Two-Phase Local Search for the Biobjective Traveling Salesman Problem. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 479–493, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[95] Vilfredo Pareto. *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne, 1896.

[96] K.E. Parsopoulos and M.N. Vrahatis. Particle Swarm Optimization Method in Multiobjective Problems. In *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC'2002)*, pages 603–607, Madrid, Spain, 2002. ACM Press.

[97] Robin Charles Purshouse. *On the Evolutionary Optimisation of Many Objectives*. PhD thesis, Department of Automatic Control and Systems Engineering, The University of Sheffield, Sheffield, UK, September 2003.

[98] Singiresu S. Rao. *Engineering Optimization*. John Wiley and Sons, third edition, 1996.

[99] Tapabrata Ray, Tai Kang, and Seow Kian Chye. An Evolutionary Algorithm for Constrained Optimization. In Darrell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2000)*, pages 771–777, San Francisco, California, 2000. Morgan Kaufmann.

[100] Tapabrata Ray and K.M. Liew. A Swarm Metaphor for Multiobjective Design Optimization. *Engineering Optimization*, 34(2):141–153, March 2002.

[101] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[102] R. S. Rosenberg. *Simulation of genetic populations with biochemical properties*. PhD thesis, University of Michigan, Ann Harbor, Michigan, 1967.

[103] Jon Rowe, Kevin Vinsen, and Nick Marvin. Parallel GAs for Multiobjective Functions. In Jarmo T. Alander, editor, *Proceedings of the Second Nordic Workshop on Genetic Algorithms and Their Applications (2NWGA)*, pages 61–70, Vaasa, Finland, August 1996. University of Vaasa.

[104] Thomas P. Runarsson and Xin Yao. Stochastic Ranking for Constrained Evolutionary Optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, September 2000.

[105] M. Sakawa. Interactive multiobjective decision making by the sequential proxy optimization technique: Spot. *European Journal of Operational research*, 9(No 4):386–396, 1982.

[106] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[107] J. David Schaffer. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pages 93–100. Lawrence Erlbaum, 1985.

[108] Jason R. Schott. Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1995.

[109] Hans-Paul Schwefel. Kybernetische evolution als strategie der experimentellen forschung inder strömungstechnik. Dipl.-Ing. thesis, 1965. (in German).

[110] N. Srinivas and Kalyanmoy Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, Fall 1994.

[111] Dipti Srinivasan and Tian Hou Seow. Particle Swarm Inspired Evolutionary Algorithm (PS-EA) for Multiobjective Optimization Problem. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 4, pages 2292–2297, Canberra, Australia, December 2003. IEEE Press.

[112] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application.* John Wiley, New York, 1986.

[113] K.C. Tan, T.H. Lee, and E.F. Khor. Evolutionary Algorithms with Dynamic Population Size and Local Exploration for Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 5(6):565–588, December 2001.

[114] Gregorio Toscano Pulido and Carlos A. Coello Coello. The Micro Genetic Algorithm 2: Towards Online Adaptation in Evolutionary Multiobjective Optimization. In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 252–266, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.

[115] Gregorio Toscano-Pulido and Carlos A. Coello Coello. A Constraint-Handling Mechanism for Particle Swarm Optimization. In *Proceedings of the Congress on Evolutionary Computation 2004 (CEC'2004)*, volume 2, pages 1396–1403, Piscataway, New Jersey, June 2004. Portland, Oregon, USA, IEEE Service Center.

[116] Frans van den Bergh. *An Analysis of Particle Swarm Optimization*. PhD thesis, Faculty of Natural and Agricultural Science, University of Petoria, Pretoria, South Africa, November 2002.

[117] David A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May 1999.

[118] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1998.

[119] David A. Van Veldhuizen and Gary B. Lamont. Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art. *Evolutionary Computation*, 8(2):125–147, 2000.

[120] David A. Van Veldhuizen and Gary B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In *2000 Congress on Evolutionary Computation*, volume 1, pages 204–211, Piscataway, New Jersey, July 2000. IEEE Service Center.

[121] A. P. Wierzbicki. Basic properties of scalarization functionals for multiobjective optimization. *Mathematische Operationsforshung and Statistick, Ser. Optimization*, 8(No. 1):55–60, 1977.

[122] A. P. Wierzbicki. The Use of Reference Objectives in Multiobjective Optimization. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making Theory and Application*, pages 469–486. Springer-Verlag, New York, 1980.

[123] August Wismann. *The Germ Plasm: A Theory of Heredity*. Scott, London, UK. United Kingdom, 1893.

[124] David H. Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[125] P. L. Yu. A class of solutions for group decision problems. *Management Science*, 19(No. 8):936–946, 1973.

[126] L. A. Zadeh. Optimality and Nonscalar-Valued Performance Criteria. *IEEE Transactions on Automatic Control*, AC-8(1):59–60, 1963.

[127] M. Zeleny. Compromise Programming. In M. K. Starr and M. Zeleny, editors, *Multiple Criteria Decision Making*. University of South Carolina Press, Columbia, South Carolina, 1973.

[128] L.B. Zhang, C.G. Zhou, X.H. Liu, Z.Q. Ma, and Y.C. Liang. Solving Multi Objective Optimization Problems Using Particle Swarm Optimization. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 4, pages 2400–2405, Canberra, Australia, December 2003. IEEE Press.

[129] Zhong-Yao Zhu and Kwong-Sak Leung. Asynchronous Self-Adjustable Island Genetic Algorithm for Multi-Objective Optimization Problems. In *Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 837–842, Piscataway, New Jersey, May 2002. IEEE Service Center.

[130] Eckart Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, November 1999.

[131] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Technical Report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, December 1999.

[132] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, Summer 2000.

[133] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.

[134] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, editors, *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2002.

[135] Eckart Zitzler, Marco Laumanns, Lothar Thiele, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Why Quality Assessment of Multiobjective Optimizers Is Difficult. In W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pages 666–673, San Francisco, California, July 2002. Morgan Kaufmann Publishers.

[136] Eckart Zitzler and Lothar Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Technical Report 43, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, May 1998.

[137] Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.