

# AN EMPIRICAL STUDY OF EVOLUTIONARY TECHNIQUES FOR MULTIOBJECTIVE OPTIMIZATION IN ENGINEERING DESIGN

AN ABSTRACT  
SUBMITTED ON THE 4TH DAY OF APRIL 1996  
TO THE DEPARTMENT OF COMPUTER SCIENCE  
OF THE GRADUATE SCHOOL OF  
TULANE UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
BY

---

CARLOS ARTEMIO COELLO COELLO

APPROVED:

---

ALAN D. CHRISTIANSEN, PH.D.  
CHAIR

---

BILL P. BUCKLES, PH.D.

---

FREDERICK E. PETRY, PH.D.

Most real-world engineering optimization problems are multiobjective in nature, since they normally have several (possibly conflicting) objectives that must be satisfied at the same time. The word “optimum” has several interpretations within this context, and it is up to the designer to decide which fits better to his/her application. Currently, there are more than 20 mathematical programming multiobjective optimization techniques, each one corresponding to a different understanding of the term “optimum”. On the other hand, genetic algorithms (GAs) have been viewed to be, since their early days, well suited for multiobjective optimization problems. Consequently, several GA-based techniques have been developed since then.

The purpose of this research has been to develop a platform that allows the testing and comparison of existing and future multiobjective optimization techniques. Two new multiobjective optimization GA-based methods based on the notion of min-max optimum are proposed, showing that at least one of them is able to produce better results than any other technique tested. Also, a method for adjusting the parameters of the GA for single-objective numerical optimization is proposed, showing the suitability of the GA as a numerical optimization technique when used properly. Then, a brief study of the importance of population policies and proper niching parameters is included. This work tries to narrow the gap between theory and practice in the context of engineering optimization. Finally, some insights on the importance of choosing a good chromosomal representation and the use of a proper fitness function are provided, derived from the analysis of a more general design problem.

# AN EMPIRICAL STUDY OF EVOLUTIONARY TECHNIQUES FOR MULTIOBJECTIVE OPTIMIZATION IN ENGINEERING DESIGN

A DISSERTATION  
SUBMITTED ON THE 4TH DAY OF APRIL 1996  
TO THE DEPARTMENT OF COMPUTER SCIENCE  
OF THE GRADUATE SCHOOL OF  
TULANE UNIVERSITY  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
BY

---

CARLOS ARTEMIO COELLO COELLO

APPROVED:

---

ALAN D. CHRISTIANSEN, PH.D.  
CHAIR

---

BILL P. BUCKLES, PH.D.

---

FREDERICK E. PETRY, PH.D.

© Copyright 1996

by

Carlos Artemio Coello Coello

All rights reserved.

# Acknowledgement

I would like to thank my parents, Carlos A. Coello Blanco and Victoria Coello de Coello for their support and encouragement to pursue graduate studies in the United States. Also, thanks to Alan D. Christiansen, my thesis advisor. His sharp mind and clear ideas always gave me the insights necessary to get into the right track. He went beyond his duties as an advisor and became a good friend, always willing to help and talk about my work and my ideas on practically any field. Thanks, Alan, for all your knowledge and your wise advice.

I also want to thank Dr. Bill P. Buckles and Dr. Fred Petry for their help and ideas, to Dr. Mike Rudnick for awakening on me an early interest on genetic algorithms and to Dr. Johnette Hassell for giving me the opportunity to pursue graduate studies in Computer Science regardless of my unsuitable background.

Finally, I also want to thank my fiancée Ma. Guadalupe Castillo Tapia for bringing light to my life in the moment of greatest darkness. Her love and support were one of my main sources of strength in some of the crucial moments during the writing of this thesis. I also want to thank Arturo Hernández Aguirre, one of major influences and most authentic friendships that I have ever had in my life. Thanks also to my sister, Delia Concepción Coello Coello, to my grandmother Consuelo Trujillo, to my deceased aunt Concepción Coello Blanco, to my aunt Flor de María Coello de Espinosa, to my aunt Elvia Coello Blanco, to my uncle

Oscar Coello Blanco, to my cousins and friends in New Orleans and México, and to all the people who gave me their moral support during all these long years.

I want to dedicate this dissertation to the four most important characters in my life:

- To my father Carlos, whom I have always admired for his multiple talents and honest way of living.
- To my mother Victoria, who is the maximum example of hard work that I have seen during my entire life.
- To my fiancée Lupita, who made me recover my faith in love when I had almost lost it.
- To my sister Delia, who will always be in my memory although time passes by and we remain far away from each other.

# Contents

<b>1 Basic Concepts</b>	<b>3</b>
<b>2 Mathematical Programming Techniques</b>	<b>28</b>
2.1 The Sequential Optimization Method . . . . .	29
2.2 The Weighting Objectives Method . . . . .	30
2.3 The $\varepsilon$ -constraint Method . . . . .	33
2.4 Global Criterion Method . . . . .	35
2.5 Goal Programming . . . . .	39
2.6 Game Theory Approach . . . . .	43
2.7 Metagames and Hypergames . . . . .	48
2.8 Multiattribute Utility Theory . . . . .	54
2.9 Surrogate Worth Trade-Off . . . . .	56
2.10 ELECTRE I and II . . . . .	58
2.11 Multicriterion Polyhedral Dynamics or Q-Analysis . . . . .	61
2.12 PROMETHEE . . . . .	62
2.13 Dynamic Compromise Programming . . . . .	64
2.14 PROTRADE . . . . .	65
2.15 STEP Method (STEM) . . . . .	66
2.16 The Method of Zions-Wallenius . . . . .	68

2.17	Sequential Multiobjective Problem Solving Method (SEMOPS) . .	71
2.18	Local Multiattribute Utility Functions . . . . .	74
2.19	The Method of Nijkamp and Vos . . . . .	76
2.20	The Nested Lagrangian Multiplier Method (NLM) . . . . .	78
2.21	Rao's Method for Fuzzy Systems . . . . .	79
2.22	Displaced Ideal . . . . .	84
2.23	Lexicographic Method . . . . .	87
2.24	Goal-Attainment Method . . . . .	88
<b>3</b>	<b>Multiobjective Optimization using Genetic Algorithms</b>	<b>92</b>
3.1	A Gentle Introduction to Genetic Algorithms . . . . .	92
3.2	Multiobjective Optimization using GAs . . . . .	104
3.3	Use of aggregating functions . . . . .	105
3.3.1	Weighted sum approach . . . . .	105
3.3.2	Reduction to a single objective . . . . .	105
3.3.3	Goal attainment . . . . .	106
3.3.4	Use of Penalty Functions . . . . .	106
3.4	Non-Pareto approaches . . . . .	107
3.4.1	VEGA . . . . .	107
3.4.2	Lexicographic ordering . . . . .	109
3.4.3	Evolutionary Strategies . . . . .	110
3.4.4	Weighted Sum . . . . .	110
3.5	Pareto-based approaches . . . . .	111
3.5.1	Pareto-based fitness assignment . . . . .	111
3.5.2	Multiple Objective Genetic Algorithm . . . . .	112



3.5.3	Non-dominated Sorting Genetic Algorithm . . . . .	113
3.5.4	Niched Pareto GA . . . . .	115
3.6	Summary of Methods . . . . .	118
<b>4</b>	<b>Implementation of MOSES</b>	<b>122</b>
4.1	Generating Pareto Optimal Solutions . . . . .	123
4.2	The min-max algorithm . . . . .	126
4.3	Monte Carlo Methods . . . . .	128
4.3.1	Monte Carlo method 1 . . . . .	129
4.3.2	Monte Carlo method 2 . . . . .	129
4.4	Osyczka's Multicriterion Optimization System . . . . .	131
4.5	Implementing GA-based approaches . . . . .	132
4.5.1	Lexicographic Method . . . . .	134
4.5.2	Schaffer's VEGA . . . . .	135
4.5.3	Hajela's Approach . . . . .	135
4.5.4	The Niched Pareto Genetic Algorithm . . . . .	136
4.5.5	The Nondominated Sorting Genetic Algorithm . . . . .	137
4.5.6	The Multiple Objective Genetic Algorithm . . . . .	138
4.5.7	An Approach Based on a Weighted Min-Max Strategy . . . . .	139
4.5.8	An Approach Based on Min-Max Selection with Sharing . . . . .	140
4.5.9	The GA optimizer for single-objective problems . . . . .	141
<b>5</b>	<b>Some Engineering Design Examples</b>	<b>144</b>
5.1	Example 1 : Design of an I-beam . . . . .	145
5.2	Example 2 : Machining recommendations . . . . .	148
5.3	Example 3 : Design of a machine tool spindle . . . . .	151

5.4	Example 4 : Design of a 10-bar plane truss . . . . .	155
5.5	Example 5 : Design of a 25-bar space truss . . . . .	155
5.6	Example 6 : Design of a 200-bar plane truss . . . . .	158
5.7	Example 7 : Design of a robot arm . . . . .	158
5.8	Example 8 : Design of a combinatorial circuit . . . . .	173
5.9	Summary and additional comments . . . . .	174
<b>6</b>	<b>Analysis of Results</b>	<b>177</b>
6.1	Measuring the Complexity of each Algorithm . . . . .	178
6.1.1	Monte Carlo Methods . . . . .	181
6.1.2	Osyczka's Multiobjective Optimization System . . . . .	182
6.1.3	Lexicographic Method and Linear Combination . . . . .	186
6.1.4	VEGA . . . . .	187
6.1.5	NSGA . . . . .	187
6.1.6	MOGA . . . . .	189
6.1.7	NPGA . . . . .	190
6.1.8	Hajela's Method . . . . .	190
6.1.9	My Weighted Min-Max Method . . . . .	191
6.1.10	My Min-Max Strategy with Sharing . . . . .	192
6.2	Example 1 : Design of an I-Beam . . . . .	192
6.3	Example 2 : Machining Recommendations . . . . .	225
6.4	Example 3 : Design of a Machine Tool Spindle . . . . .	230
6.5	Example 4 : Design of a 10-bar Plane Truss . . . . .	258
6.6	Example 5 : Design of a 25-bar Space Truss . . . . .	295
6.7	Example 6 : Design of a 200-bar Plane Truss . . . . .	327

6.8	Example 7 : Design of a Robot Arm . . . . .	367
6.9	Example 8 : Design of a Combinatorial Circuit . . . . .	372
6.10	Critique of each Method . . . . .	377
6.10.1	Monte Carlo Method 1 . . . . .	377
6.10.2	Monte Carlo Method 2 . . . . .	378
6.10.3	Osyczka's Multiobjective Optimization System . . . . .	379
6.10.4	GA-based Linear Combination of Objectives . . . . .	380
6.10.5	Lexicographic Method . . . . .	380
6.10.6	VEGA . . . . .	381
6.10.7	NSGA . . . . .	381
6.10.8	MOGA . . . . .	382
6.10.9	NPGA . . . . .	383
6.10.10	Hajela's Method . . . . .	384
6.10.11	GA with a Weighted Min-Max Technique . . . . .	385
6.10.12	GA with Min-Max Binary Tournament Selection and Sharing	386
<b>7</b>	<b>Discussion</b>	<b>388</b>
7.1	Population Policies . . . . .	389
7.2	Niching Parameters . . . . .	395
7.3	Incorporating Knowledge About the Domain . . . . .	405
7.4	An Expert System to Select a Multiobjective Optimization Method	408
<b>8</b>	<b>Conclusions</b>	<b>411</b>
8.1	Contributions . . . . .	411
8.2	Future Work . . . . .	416



# List of Tables

5.1	Machinability data for 390 die cast/carbonide/wet. . . . .	148
5.2	Loading conditions for the 25-bar space truss shown in Figure 5.5.	156
5.3	Group membership for the 25-bar space truss shown in Figure 5.5.	157
5.4	Coordinates of the joints of the 25-bar space truss shown in Figure 5.5. . . . .	157
5.5	Group membership for the 200-bar plane truss shown in Figure 5.6.	159
6.1	Comparison of results computing the ideal vector of example 1 from Chapter 5 (design of an I-beam). For each method the best results for optimum $f_1$ and $f_2$ are shown in <b>boldface</b> . OS stands for Osyczka's Multiobjective Optimization System. . . . .	193

6.2	Comparison of the best overall solution found by each one of the methods included in MOSES for the first example. GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.5 (equal weight for both objectives).	198
6.3	Comparison of results computing the ideal vector of example 2 from Chapter 5 (machining recommendations). For each method the best results for each objective function are shown in <b>boldface</b> . OS stands for Osyczka's Multiobjective Optimization System.	226
6.4	(Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the second example (machining recommendations). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective). (Continued in Table 6.5)	227

6.5 (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the second example (machining recommendations). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective). . . . .	228
6.6 Comparison of results computing the ideal vector of example 3 from Chapter 5 (design of a machine tool spindle). For each method the best results for optimum $f_1$ and $f_2$ are shown in <b>boldface</b> . . . . .	230
6.7 Comparison of the best overall solution found by each one of the methods included in MOSES for the third example (design of a machine tool spindle). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.5 (equal weight for every objective). . . . .	237
6.8 (Part I) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). For each method the best results for optimum $f_1$ , $f_2$ and $f_3$ are shown in <b>boldface</b> . OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.9, 6.10 and 6.11) . . . . .	259

6.9 (Part II) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.10 and 6.11) . . . . .	259
6.10 (Part III) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.11) . . . . .	260
6.11 (Part IV) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. . . . .	260
6.12 (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.13, 6.14 and 6.15) . . . . .	269



- 6.13 (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.14 and 6.15) . . . . . 270
- 6.14 (Part III) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Table 6.15)271

6.15 (Part IV) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). . . . .	272
6.16 (Part I) Comparison of results computing the ideal vector of example 5 from Chapter 5 (design of a 25-bar space truss). For each method the best results for optimum $f_1$ , $f_2$ and $f_3$ are shown in <b>boldface</b> . OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.17) . . . . .	296
6.17 (Part II) Comparison of results computing the ideal vector of example 5 from Chapter 5 (design of a 25-bar space truss). OS stands for Osyczka's Multiobjective Optimization System. . . . .	296

6.18 (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the fifth example (design of a 25-bar space truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Table 6.19)	304
6.19 (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the fifth example (design of a 25-bar space truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). . . . .	305
6.20 (Part I) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). For each method the best results for optimum $f_1$ , $f_2$ and $f_3$ are shown in <b>boldface</b> . OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.21, 6.22, 6.23 and 6.24) . . . . .	328

6.21	(Part II) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.22, 6.23 and 6.24) . . . . .	329
6.22	(Part III) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.23 and 6.24) . . . . .	329
6.23	(Part IV) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.24) . . . . .	330
6.24	(Part V) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. . . . .	330

6.25 (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.26, 6.27, 6.28 and 6.29) . . . . .	339
6.26 (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.27, 6.28 and 6.29) . . . . .	340

- 6.27 (Part III) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.28 and 6.29) . . . . . 341
- 6.28 (Part IV) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Table 6.29) 342

6.29	(Part V) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). . . . .	343
6.30	(Part I) Comparison of results computing the ideal vector of example 7 from Chapter 5 (design of a robot arm). For each method the best results for optimum $f_1$ , $f_2$ , $f_3$ and $f_4$ are shown in <b>bold-face</b> . OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.31) . . . . .	367
6.31	(Part II) Comparison of results computing the ideal vector of example 7 from Chapter 5 (design of a robot arm). OS stands for Osyczka's Multiobjective Optimization System. . . . .	368

6.32 (Part I)	Comparison of the best overall solution found by each one of the methods included in MOSES for the seventh example (design of a robot arm). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective). (Continued in Table 6.33)	370
6.33 (Part II)	Comparison of the best overall solution found by each one of the methods included in MOSES for the seventh example (design of a robot arm). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective). . . . .	371



7.1	Comparison of results using different population sizes with my min-max method that uses sharing. $M$ is the population size, $L_p(f)$ is the maximum deviation with respect to the ideal vector and $E$ is the number of function evaluations required. These results correspond to example 1 from Chapter 5 using binary (B) and floating point (FP) representations. . . . .	390
7.2	Comparison of results using different population sizes with my min-max method that uses sharing. $M$ is the population size, $L_p(f)$ is the maximum deviation with respect to the ideal vector and $E$ is the number of function evaluations required. These results correspond to example 2 from Chapter 5 using binary (B) and floating point (FP) representations. . . . .	392
7.3	Comparison of results using different population sizes with my min-max method that uses sharing. $M$ is the population size, $L_p(f)$ is the maximum deviation with respect to the ideal vector and $E$ is the number of function evaluations required. These results correspond to example 3 from Chapter 5 using binary (B) and floating point (FP) representations. . . . .	393
7.4	Comparison of results using different values of $\sigma_{share}$ with my min-max method that uses sharing. $L_p(f)$ is the maximum deviation with respect to the ideal vector. These results correspond to example 1 from Chapter 5 using binary (B) and floating point (FP) representations. . . . .	396

7.5	Comparison of results using different values of $\sigma_{share}$ with my min-max method that uses sharing. $L_p(f)$ is the maximum deviation with respect to the ideal vector. These results correspond to example 2 from Chapter 5 using binary (B) and floating point (FP) representations. The asterisk (*) indicates total convergence of the population to a unique solution. . . . .	396
7.6	Comparison of results using different values of $\sigma_{share}$ with my min-max method that uses sharing. $L_p(f)$ is the maximum deviation with respect to the ideal vector. These results correspond to example 3 from Chapter 5 using binary (B) and floating point (FP) representations. . . . .	397
7.7	Comparison of results using estimated values of $\sigma_{share}$ with my min-max method. $L_p(f)$ is the maximum deviation with respect to the ideal vector. The value of $q$ indicated the number of desired peaks to which we want the population to converge. . . . .	402

# List of Figures

1.1	Ideal solution in which all our functions have their minimum at a common point. . . . .	9
1.2	Two examples of convex sets. . . . .	10
1.3	Two examples of non-convex sets. . . . .	10
1.4	Graphical representation of the $t$ -directional shadow for the range $F$ . . . . .	11
1.5	Weakly and strongly non-dominated curves on the biobjective case. . . . .	12
1.6	An example of a problem with two variables and two objective functions. The pareto optimal solutions are indicated by the shaded boundaries of the design region. . . . .	17
1.7	A three-bar plane truss used to illustrate the basic concepts covered in this chapter. . . . .	18
1.8	Euclidean space of the decision variables for the three-bar truss of Figure 1.7. The curve $g_1$ limits the feasible region. . . . .	20
1.9	Design region for the three-bar plane truss of Figure 1.7. . . . .	22
1.10	The solution region for the three-bar plane truss of Figure 1.7 is delimited by points $F(x_a)$ , $F(x_b)$ , $F(x_c)$ , $F(x_d)$ and $F(x_e)$ . The point $F(x_f)$ corresponds to the solution adopted. . . . .	25

1.11	The normalized solution region for the three-bar plane truss of Figure 1.7 is delimited by points $F(x_a)$ , $F(x_b)$ , $F(x_c)$ , $F(x_d)$ and $F(x_e)$ . The point $F(\text{Gamma})$ corresponds to the solution adopted. . . . .	26
2.1	The weighting objectives method for a maximizing problem. . . .	31
2.2	The $\varepsilon$ -constraint method for a maximizing problem. . . . .	34
2.3	Sketch of a compromise solution. The basic idea is to take the point which is closest, by some distance measure, to the ideal point, which in this case is the origin. . . . .	37
2.4	Example of cooperative and non-cooperative game solutions. . . .	42
2.5	Preferences for player A. . . . .	49
2.6	Generalized preferences for player A. . . . .	50
2.7	Stability analysis of a particular outcome $s$ for a particular player $i$ . .	51
2.8	Example of an ELECTRE graph. Each node corresponds to a non-dominated alternative. The arrows indicate preferences. Therefore we can say that alternative 1 is preferred to alternative 5, alternative 4 is preferred to alternative 6, etc. . . . .	60
2.9	An example of a case where reaching the ideal point ( $M$ ) is an unrealistic goal, and we search, instead, an alternative point ( $I$ ). .	85
2.10	Goal-attainment method with two objective functions. . . . .	90
3.1	Use of a single-point crossover between two chromosomes. Notice that each pair of chromosomes produces two descendants for the next generation. The cross-point may be located at the string boundaries, in which case the crossover has no effect and the parents remain intact for the next generation. . . . .	100

3.2	Use of a two-point crossover between two chromosomes. In this case the genes at the extremes are kept, and those in the middle part are exchanged. If one of the two cross-points happens to be at the string boundaries, a single-point crossover will be performed, and if both are at the string boundaries, the parents remain intact for the next generation. . . . .	100
3.3	Representing the same number using binary and floating point encodings. . . . .	102
3.4	Schematic of VEGA selection. . . . .	107
3.5	Flowchart of the Nondominated Sorting Genetic Algorithm (NSGA). . . . .	114
4.1	Graphical illustration of the contact theorem. . . . .	124
4.2	Graphical illustration of equations (4.3) and (4.4). . . . .	125
5.1	The simply supported I-beam of Example 1. . . . .	145
5.2	Sketch of the machine tool spindle used for Example 3. . . . .	151
5.3	10-bar plane truss used for Example No. 4. . . . .	154
5.4	Cross-section used for Example No. 4. . . . .	154
5.5	25-bar space truss used for example No. 5. . . . .	156
5.6	200-bar plane truss used for example No. 6. . . . .	160
5.7	PUMA-560 robot arm and schematic representation of coordinate angles $\theta_i$ . . . . .	161
5.8	Mechanical model of the robot arm used for optimization. . . . .	163
5.9	Free-body diagrams of the robot arm. . . . .	166
5.10	Angular velocities and corresponding angular accelerations of the robot arm used for Example No. 8. . . . .	170

5.11	A gate in a two-dimensional template, gets its second input from either one of two gates in the previous column. . . . .	173
6.1	Example 1: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed. . . . .	193
6.2	Example 1: Initial feasible region for Monte Carlo method. . . . .	194
6.3	Example 1: Initial feasible region. . . . .	194
6.4	Example 1: The GA using a linear combination of the objectives with scaling, after 50 generations. . . . .	196
6.5	Example 1: Distribution of points using VEGA with a binary representation at generation twenty. . . . .	199
6.6	Example 1: Distribution of points using VEGA with a binary representation at generation fifty. . . . .	199
6.7	Example 1: Distribution of points using VEGA with floating point representation at generation twenty. . . . .	200
6.8	Example 1: Distribution of points using VEGA with floating point representation at generation fifty. . . . .	200
6.9	Example 1: Distribution of points using VEGA with binary representation at generation 100. . . . .	201
6.10	Example 1: Distribution of points using NSGA with binary representation at generation fifty. . . . .	202
6.11	Example 1: Distribution of points using NSGA with binary representation at generation twenty. . . . .	203
6.12	Example 1: Distribution of points using NSGA with binary representation at generation 500. . . . .	203

6.13	Example 1: Distribution of points using NSGA with floating point representation at generation 50. . . . .	204
6.14	Example 1: Distribution of points using MOGA with binary representation at generation 20. . . . .	205
6.15	Example 1: Distribution of points using MOGA with binary representation at generation 50. . . . .	206
6.16	Example 1: Distribution of points using MOGA with binary representation at generation 100. . . . .	206
6.17	Example 1: Distribution of points using MOGA with floating point representation at generation 50. . . . .	207
6.18	Example 1: Distribution of points using NPGA with binary representation at generation 20. . . . .	208
6.19	Example 1: Distribution of points using NPGA with binary representation at generation 50. . . . .	209
6.20	Example 1: Distribution of points using NPGA with binary representation at generation 100. . . . .	209
6.21	Example 1: Distribution of points using NPGA with binary representation at generation 500. . . . .	210
6.22	Example 1: Distribution of points using NPGA with binary representation at generation 50 with $\sigma_{share} = 1.0$ . . . . .	210
6.23	Example 1: Distribution of points using NPGA with floating point representation at generation 50. . . . .	211
6.24	Example 1: Distribution of points using NPGA with floating point representation at generation 100. . . . .	211

6.25	Example 1: Distribution of points using Hajela's method with binary representation at generation 20. . . . .	213
6.26	Example 1: Distribution of points using Hajela's method with binary representation at generation 50. . . . .	214
6.27	Example 1: Distribution of points using Hajela's method with binary representation at generation 50 using 500 individuals. . . .	214
6.28	Example 1: Distribution of points using Hajela's method with floating point representation at generation 50. . . . .	215
6.29	Example 1: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero. . . .	216
6.30	Example 1: Distribution of points using my method based on the min-max algorithm with binary representation at generation 50. . .	217
6.31	Example 1: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 50. . . . .	217
6.32	Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20. . . . .	219
6.33	Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 50. . . . .	219
6.34	Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100. . . . .	220



6.35	Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 500. . . . .	220
6.36	Example 1: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20. . . . .	221
6.37	Example 1: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 50. . . . .	221
6.38	Example 1: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. . . . .	223
6.39	Example 1: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. . . . .	224
6.40	Example 1: Distribution of points using the Lexicographic Method with a binary representation at generation fifty. . . . .	224
6.41	Example 1: Distribution of points using the Lexicographic Method with a floating point representation at generation fifty. . . . .	225
6.42	Example 3: Initial feasible region for example 3. . . . .	231
6.43	Example 3: Initial feasible region for Monte Carlo method solving the third example. . . . .	231
6.44	Example 3: The GA using a linear combination of the objectives with scaling, after 50 generations using binary representation. . .	232
6.45	Example 3: The GA using a linear combination of the objectives with scaling, after 50 generations using floating point representation.	233

6.46	Example 3: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed. . . . .	234
6.47	Example 3: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed. . . . .	234
6.48	Example 3: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed. . . . .	235
6.49	Example 3: Distribution of points using the Lexicographic Method with a binary representation at generation fifty. Only points within the feasible region are displayed. . . . .	235
6.50	Example 3: Distribution of points using the Lexicographic Method with a floating point representation at generation fifty. Only points within the feasible region are displayed. . . . .	236
6.51	Example 3: Distribution of points using VEGA with a binary representation at generation twenty. . . . .	238
6.52	Example 3: Distribution of points using VEGA with a binary representation at generation fifty. . . . .	238
6.53	Example 3: Distribution of points using VEGA with floating point representation at generation twenty. . . . .	239
6.54	Example 3: Distribution of points using VEGA with floating point representation at generation fifty. . . . .	239
6.55	Example 3: Distribution of points using VEGA with binary representation at generation 100. . . . .	240

6.56	Example 3: Distribution of points using NSGA with binary representation at generation fifty. . . . .	241
6.57	Example 3: Distribution of points using NSGA with binary representation at generation twenty. . . . .	242
6.58	Example 3: Distribution of points using NSGA with binary representation at generation 500. . . . .	242
6.59	Example 3: Distribution of points using NSGA with floating point representation at generation 50. . . . .	243
6.60	Example 3: Distribution of points using MOGA with binary representation at generation 20. . . . .	244
6.61	Example 3: Distribution of points using MOGA with binary representation at generation 50. . . . .	244
6.62	Example 3: Distribution of points using MOGA with binary representation at generation 100. . . . .	245
6.63	Example 3: Distribution of points using MOGA with floating point representation at generation 50. . . . .	245
6.64	Example 3: Distribution of points using NPGA with binary representation at generation 20. . . . .	246
6.65	Example 3: Distribution of points using NPGA with binary representation at generation 50. . . . .	247
6.66	Example 3: Distribution of points using NPGA with binary representation at generation 100. . . . .	247
6.67	Example 3: Distribution of points using NPGA with binary representation at generation 500. . . . .	248

6.68	Example 3: Distribution of points using NPGA with binary representation at generation 50 with $\sigma_{share} = 1.0$ . . . . .	248
6.69	Example 3: Distribution of points using NPGA with floating point representation at generation 50. . . . .	249
6.70	Example 3: Distribution of points using NPGA with floating point representation at generation 20. . . . .	249
6.71	Example 3: Distribution of points using Hajela's method with binary representation at generation 20. . . . .	250
6.72	Example 3: Distribution of points using Hajela's method with binary representation at generation 50. . . . .	251
6.73	Example 3: Distribution of points using Hajela's method with binary representation at generation 50 using 500 individuals. . . .	251
6.74	Example 3: Distribution of points using Hajela's method with floating point representation at generation 50. . . . .	252
6.75	Example 3: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero. . . .	253
6.76	Example 3: Distribution of points using my method based on the min-max algorithm with binary representation at generation 50. . .	253
6.77	Example 3: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 50. . . . .	254
6.78	Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20. . . . .	255

6.79	Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 50. . . . .	255
6.80	Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100. . . . .	256
6.81	Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 500. . . . .	256
6.82	Example 3: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20. . . . .	257
6.83	Example 3: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 50. . . . .	257
6.84	Initial feasible region for Monte Carlo method solving the fourth example. . . . .	261
6.85	Example 4: The GA using a linear combination of the objectives with scaling, after 100 generations using binary representation. . .	263
6.86	Example 4: The GA using a linear combination of the objectives with scaling, after 100 generations using floating point representation.	263
6.87	Example 4: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed. . . . .	264

6.88	Example 4: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed. . . . .	265
6.89	Example 4: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed. . . . .	265
6.90	Example 4: Distribution of points using the Lexicographic Method with a binary representation at generation 100. Only points within the feasible region are displayed. . . . .	266
6.91	Example 4: Distribution of points using the Lexicographic Method with a floating point representation at generation 100. Only points within the feasible region are displayed. . . . .	266
6.92	Example 4: Distribution of points using VEGA with a binary representation at generation twenty. . . . .	268
6.93	Example 4: Distribution of points using VEGA with a binary representation at generation fifty. . . . .	273
6.94	Example 4: Distribution of points using VEGA with floating point representation at generation twenty. . . . .	273
6.95	Example 4: Distribution of points using VEGA with floating point representation at generation fifty. . . . .	274
6.96	Example 4: Distribution of points using VEGA with binary representation at generation 100. . . . .	274
6.97	Example 4: Distribution of points using VEGA with floating point representation at generation 100. . . . .	275

6.98	Example 4: Distribution of points using NSGA with binary representation at generation 100. . . . .	276
6.99	Example 4: Distribution of points using NSGA with binary representation at generation twenty. . . . .	276
6.100	Example 4: Distribution of points using NSGA with binary representation at generation 500. . . . .	277
6.101	Example 4: Distribution of points using NSGA with floating point representation at generation 100. . . . .	277
6.102	Example 4: Distribution of points using MOGA with binary representation at generation 20. . . . .	279
6.103	Example 4: Distribution of points using MOGA with binary representation at generation 100. . . . .	279
6.104	Example 4: Distribution of points using MOGA with binary representation at generation 500. . . . .	280
6.105	Example 4: Distribution of points using MOGA with floating point representation at generation 100. . . . .	280
6.106	Example 4: Distribution of points using NPGA with binary representation at generation 20. . . . .	281
6.107	Example 4: Distribution of points using NPGA with binary representation at generation 50. . . . .	282
6.108	Example 4: Distribution of points using NPGA with binary representation at generation 100. . . . .	282
6.109	Example 4: Distribution of points using NPGA with binary representation at generation 500. . . . .	283

6.110	Example 4: Distribution of points using NPGA with binary representation at generation 100 with $\sigma_{share} = 1.0$ . . . . .	283
6.111	Example 4: Distribution of points using NPGA with floating point representation at generation 100. . . . .	284
6.112	Example 4: Distribution of points using NPGA with floating point representation at generation 20. . . . .	284
6.113	Example 4: Distribution of points using Hajela's method with binary representation at generation 20. . . . .	286
6.114	Example 4: Distribution of points using Hajela's method with binary representation at generation 50. . . . .	286
6.115	Example 4: Distribution of points using Hajela's method with binary representation at generation 100. . . . .	287
6.116	Example 4: Distribution of points using Hajela's method with binary representation at generation 500 . . . . .	287
6.117	Example 4: Distribution of points using Hajela's method with floating point representation at generation 20. . . . .	288
6.118	Example 4: Distribution of points using Hajela's method with floating point representation at generation 100. . . . .	288
6.119	Example 4: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero. . . . .	290
6.120	Example 4: Distribution of points using my method based on the min-max algorithm with binary representation at generation 100. . . . .	290
6.121	Example 4: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 100. . . . .	291



6.122	Example 4: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20. . . . .	292
6.123	Example 4: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 50. . . . .	292
6.124	Example 4: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100. . . . .	293
6.125	Example 4: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20. . . . .	293
6.126	Example 4: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 100. . . . .	294
6.127	Initial feasible region for Monte Carlo method solving the fifth example. . . . .	297
6.128	Example 5: The GA using a linear combination of the objectives with scaling, after 100 generations using binary representation. . .	298
6.129	Example 5: The GA using a linear combination of the objectives with scaling, after 100 generations using floating point representation.	299
6.130	Example 5: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed. . . . .	300

6.131	Example 5: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed. . . . .	300
6.132	Example 5: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed. . . . .	301
6.133	Example 5: Distribution of points using the Lexicographic Method with a binary representation at generation 100. Only points within the feasible region are displayed. . . . .	301
6.134	Example 5: Distribution of points using the Lexicographic Method with a floating point representation at generation 100. Only points within the feasible region are displayed. . . . .	302
6.135	Example 5: Distribution of points using VEGA with a binary representation at generation twenty. . . . .	303
6.136	Example 5: Distribution of points using VEGA with a binary representation at generation fifty. . . . .	306
6.137	Example 5: Distribution of points using VEGA with floating point representation at generation twenty. . . . .	306
6.138	Example 5: Distribution of points using VEGA with floating point representation at generation fifty. . . . .	307
6.139	Example 5: Distribution of points using VEGA with binary representation at generation 100. . . . .	307
6.140	Example 5: Distribution of points using VEGA with floating point representation at generation 100. . . . .	308

6.141	Example 5: Distribution of points using NSGA with binary representation at generation 100. . . . .	309
6.142	Example 5: Distribution of points using NSGA with binary representation at generation twenty. . . . .	310
6.143	Example 5: Distribution of points using NSGA with binary representation at generation 500. . . . .	310
6.144	Example 5: Distribution of points using NSGA with floating point representation at generation 100. . . . .	311
6.145	Example 5: Distribution of points using MOGA with binary representation at generation 20. . . . .	312
6.146	Example 5: Distribution of points using MOGA with binary representation at generation 100. . . . .	312
6.147	Example 5: Distribution of points using MOGA with binary representation at generation 500. . . . .	313
6.148	Example 5: Distribution of points using MOGA with floating point representation at generation 100. . . . .	313
6.149	Example 5: Distribution of points using NPGA with binary representation at generation 20. . . . .	315
6.150	Example 5: Distribution of points using NPGA with binary representation at generation 50. . . . .	315
6.151	Example 5: Distribution of points using NPGA with binary representation at generation 100. . . . .	316
6.152	Example 5: Distribution of points using NPGA with binary representation at generation 500. . . . .	316

6.153	Example 5: Distribution of points using NPGA with binary representation at generation 100 with $\sigma_{share} = 1.0$ . . . . .	317
6.154	Example 5: Distribution of points using NPGA with floating point representation at generation 100. . . . .	317
6.155	Example 5: Distribution of points using NPGA with floating point representation at generation 20. . . . .	318
6.156	Example 5: Distribution of points using Hajela's method with binary representation at generation 20. . . . .	319
6.157	Example 5: Distribution of points using Hajela's method with binary representation at generation 50. . . . .	320
6.158	Example 5: Distribution of points using Hajela's method with binary representation at generation 100. . . . .	320
6.159	Example 5: Distribution of points using Hajela's method with binary representation at generation 500 . . . . .	321
6.160	Example 5: Distribution of points using Hajela's method with floating point representation at generation 20. . . . .	321
6.161	Example 5: Distribution of points using Hajela's method with floating point representation at generation 100. . . . .	322
6.162	Example 5: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero. . . . .	323
6.163	Example 5: Distribution of points using my method based on the min-max algorithm with binary representation at generation 100. . . . .	323
6.164	Example 5: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 100. . . . .	324

6.165	Example 5: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20. . . . .	325
6.166	Example 5: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20. . . . .	326
6.167	Example 5: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 50. . . . .	326
6.168	Example 5: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 100. . . . .	327
6.169	Initial feasible region for Monte Carlo method solving the sixth example. . . . .	331
6.170	Example 6: The GA using a linear combination of the objectives with scaling, after 100 generations using binary representation. . .	333
6.171	Example 6: The GA using a linear combination of the objectives with scaling, after 100 generations using floating point representation.	334
6.172	Example 6: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed. . . . .	335
6.173	Example 6: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed. . . . .	335

6.174	Example 6: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed. . . . .	336
6.175	Example 6: Distribution of points using the Lexicographic Method with a binary representation at generation 100. Only points within the feasible region are displayed. . . . .	336
6.176	Example 6: Distribution of points using the Lexicographic Method with a floating point representation at generation 100. Only points within the feasible region are displayed. . . . .	337
6.177	Example 6: Distribution of points using VEGA with a binary representation at generation twenty. . . . .	344
6.178	Example 6: Distribution of points using VEGA with a binary representation at generation fifty. . . . .	344
6.179	Example 6: Distribution of points using VEGA with floating point representation at generation twenty. . . . .	345
6.180	Example 6: Distribution of points using VEGA with floating point representation at generation fifty. . . . .	345
6.181	Example 6: Distribution of points using VEGA with binary representation at generation 100. . . . .	346
6.182	Example 6: Distribution of points using VEGA with floating point representation at generation 100. . . . .	346
6.183	Example 6: Distribution of points using NSGA with binary representation at generation twenty. . . . .	347
6.184	Example 6: Distribution of points using NSGA with binary representation at generation 50. . . . .	348

6.185	Example 6: Distribution of points using NSGA with binary representation at generation 100. . . . .	348
6.186	Example 6: Distribution of points using NSGA with floating point representation at generation 20. . . . .	349
6.187	Example 6: Distribution of points using NSGA with floating point representation at generation 50. . . . .	349
6.188	Example 6: Distribution of points using NSGA with floating point representation at generation 100. . . . .	350
6.189	Example 6: Distribution of points using MOGA with binary representation at generation 20. . . . .	351
6.190	Example 6: Distribution of points using MOGA with binary representation at generation 100. . . . .	352
6.191	Example 6: Distribution of points using MOGA with floating point representation at generation 20. . . . .	352
6.192	Example 6: Distribution of points using MOGA with floating point representation at generation 100. . . . .	353
6.193	Example 6: Distribution of points using NPGA with binary representation at generation 20. . . . .	354
6.194	Example 6: Distribution of points using NPGA with binary representation at generation 50. . . . .	354
6.195	Example 6: Distribution of points using NPGA with binary representation at generation 100. . . . .	355
6.196	Example 6: Distribution of points using NPGA with binary representation at generation 100 with $\sigma_{share} = 1.0$ . . . . .	355

6.197	Example 6: Distribution of points using NPGA with floating point representation at generation 100. . . . .	356
6.198	Example 6: Distribution of points using NPGA with floating point representation at generation 20. . . . .	356
6.199	Example 6: Distribution of points using NPGA with floating point representation at generation 50. . . . .	357
6.200	Example 6: Distribution of points using Hajela's method with binary representation at generation 20. . . . .	358
6.201	Example 6: Distribution of points using Hajela's method with binary representation at generation 50. . . . .	359
6.202	Example 6: Distribution of points using Hajela's method with binary representation at generation 100. . . . .	359
6.203	Example 6: Distribution of points using Hajela's method with floating point representation at generation 20. . . . .	360
6.204	Example 6: Distribution of points using Hajela's method with floating point representation at generation 20. . . . .	360
6.205	Example 6: Distribution of points using Hajela's method with floating point representation at generation 100. . . . .	361
6.206	Example 6: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero. . . . .	362
6.207	Example 6: Distribution of points using my method based on the min-max algorithm with binary representation at generation 100. . . . .	363
6.208	Example 6: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 100. . . . .	363



6.209	Example 6: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20. . . . .	364
6.210	Example 6: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100. . . . .	365
6.211	Example 6: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20. . . . .	365
6.212	Example 6: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 100. . . . .	366
6.213	Convergence history of the GA solving problem No. 8 (binary representation). The evolution of the maximum fitness is displayed through 100 generations. . . . .	374
6.214	Convergence history of the GA solving problem No. 8 (binary representation). The evolution of the average fitness is displayed through 100 generations. . . . .	375
6.215	Example 8: A sample circuit design produced by MOSES using binary representation. . . . .	376
6.216	Example 8: A sample circuit design produced by MOSES using binary representation. . . . .	376
6.217	Example 8: A sample circuit design produced by the classical genetic algorithm with an elitist selection strategy used by Sushil Louis. . . . .	377

7.1	Relation between population size and maximum fitness for the example No. 8 . . . . .	389
7.2	Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 0.0001 under binary representation. . . . .	397
7.3	Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 0.0001 under floating point representation. . . . .	398
7.4	Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 100.0 under binary representation. . . . .	398
7.5	Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 100.0 under floating point representation. . . . .	399
7.6	Distribution of points in the objective function domain for example 3 using my min-max strategy with a sharing factor of 0.0001 under binary representation. . . . .	399
7.7	Distribution of points in the objective function domain for example 3 using my min-max strategy with a sharing factor of 0.0001 under floating point representation. . . . .	400
7.8	Distribution of points in the objective function domain for example 3 using my min-max strategy with a sharing factor of 100.0 under binary representation. . . . .	400

7.9 Distribution of points in the objective function domain for example	
3 using my min-max strategy with a sharing factor of 100.0 under	
floating point representation. . . . .	401

# Introduction

Although much research has been done in engineering optimization recently, the trend has been to deal with ideal and unrealistic problems, rather than than real-world applications. One of the reasons for this has been that for many years, engineering optimization considered only single-objective functions. This is normally not a realistic assumption, given the fact that most real-world engineering problems have several (possibly conflicting) objectives. For example, if we refer to the optimization of a certain structure, we will normally want to minimize its weight and volume of material used, but at the same time, we will want to maximize its frequency, so that it can provide the maximum possible safety. These objectives are, however, conflicting, since a maximum frequency value will require a higher volume of material and, consequently, a higher weight. Currently, much work has been done in this area, and there are more than 20 mathematical optimization methods to deal with multiple objectives. In an early stage of their development, it was recognized that genetic algorithms (GAs) were well suited for multiobjective optimization problems, and several approaches were suggested.

This dissertation provides a general view of several mathematical programming approaches which can be useful in engineering design, and a comparison of these methods to GA approaches in several engineering design problems. Also, some hybrid techniques are proposed, and an empirical study of their behavior

is presented, in order to better understand when to use each one in real-world situations. With this goal in mind, a system has been developed to include all techniques of interest, and a complete analysis of their behavior under several engineering design optimization problems is provided here. Finally, some guidelines are given to suggest possible future paths of research.

The organization of this thesis is the following: Chapter 1 contains the basic concepts from operations research that are necessary to understand the material of the further chapters. Chapter 2 gives a classification and brief description of the main mathematical programming techniques for multiobjective optimization, together with some comments on their advantages and disadvantages, and where to find more information on them. Chapter 3 provides a historical review of the different approaches taken in the GA community to deal with multiple objectives. Chapter 4 describes MOSES (Multiobjective Optimization of Systems in the Engineering Sciences), its capabilities, limitations, and implementation details. Chapter 5 includes complete descriptions of several optimization engineering design problems used to test the system. Chapter 6 contains a comparison of the various methods available in terms of computational efficiency, limitations and ease of use, together with a brief theoretical analysis of the behavior of each technique in terms of the eight problems studied. Chapter 7 provides some of my experiences regarding population policies and sharing parameters, in an attempt to clarify the search for better ways to adjust the parameters of the genetic algorithm in numerical optimization problems with several objectives. Also, a brief description of an expert system that was developed to be used as a front end for MOSES is provided. Finally, the conclusions derived from this work and some possible future paths of research are given.

# Chapter 1

## Basic Concepts

Multiobjective optimization (also called multicriteria optimization, multiperformance or vector optimization) can be defined as the problem of finding [1]:

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 designer.

Most of the definitions and notation introduced in this chapter, follow that of Osyczka [2].

We will call **decision variables** the numerical quantities for which values are to be chosen in an optimization problem. These quantities will be denoted as  $x_j$ ,  $j = 1, 2, \dots, n$ .

The vector of  $n$  decision variables  $\bar{x}$  will be represented by:

$$\bar{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (1.1)$$

Also, it can be written in a more convenient way as:

$$\bar{x} = [x_1, x_2, \dots, x_n]^T, \quad (1.2)$$

where  $T$  indicates the transposition of the column vector to the row vector.

In every engineering problem there are always restrictions imposed by the particular characteristics of the environment or resources available. These restrictions must be satisfied in order to consider that a certain solution is acceptable. We will call **constraints** all these restrictions in general, which describe dependences among decision variables and constants (or parameters) involved in the problem. These constraints will be expressed in form of mathematical inequalities:

$$g_i \geq 0 \quad i = 1, \dots, m \quad (1.3)$$

or equalities:

$$h_i = 0 \quad i = 1, \dots, p \quad (1.4)$$

Note that  $p$ , the number of equality constraints, must be less than  $n$ , the number of decision variables, because if  $p \geq n$  the problem is said to be **overconstrained**, since there are no degrees of freedom left for optimizing. The

number of degrees of freedom is given by  $n - p$ . Also, constraints can be explicit or implicit, in which case the algorithm to compute  $g_i(\bar{x})$  for any given vector  $\bar{x}$  must be known.

In order to know how “good” a certain solution is, we need to have some criteria to evaluate it. These criteria are expressed as computable functions of the decision variables, that are called **objective functions**. In our case, some of them will be in conflict with others, and some will have to be minimized while others are maximized. These objective functions may be **commensurable** (measured in the same units) or **non-commensurable** (measured in different units). In general, the objective functions with which we deal in engineering optimization are non-commensurable.

We will designate the objective functions as:  $f_1(\bar{x}), f_2(\bar{x}), \dots, f_k(\bar{x})$ . Therefore, our objective functions will form a vector function  $\bar{f}(\bar{x})$  which will be defined by:

$$\bar{f}(\bar{x}) = \begin{bmatrix} f_1(\bar{x}) \\ f_2(\bar{x}) \\ \vdots \\ f_k(\bar{x}) \end{bmatrix} \quad (1.5)$$

Also, it can be written in a more convenient way as:

$$\bar{f}(\bar{x}) = [f_1(\bar{x}), f_2(\bar{x}), \dots, f_k(\bar{x})]^T \quad (1.6)$$

The set of all  $n$ -tuples of real numbers denoted by  $R^n$  is called **Euclidean  $n$ -space**. We will consider two Euclidean spaces:

- The  $n$ -dimensional space of the decision variables in which each coordinate axis corresponds to a component of vector  $\bar{x}$ .



- The  $k$ -dimensional space of the objective functions in which each coordinate axis corresponds to a component vector  $\bar{f}(\bar{x})$ .

Every point in the first space represents a solution and gives a certain point in the second space, which determines a quality of this solution in terms of the values of the objective functions.

The **multiobjective optimization problem** can now be defined as follows:

Find the vector  $\bar{x}^* = [x_1^*, x_2^*, \dots, x_n^*]^T$  which will satisfy the  $m$  inequality constraints:

$$g_i(\bar{x}) \geq 0 \quad i = 1, 2, \dots, m \quad (1.7)$$

the  $p$  equality constraints

$$h_i(\bar{x}) = 0 \quad i = 1, 2, \dots, p \quad (1.8)$$

and optimize the vector function

$$\bar{f}(\bar{x}) = [f_1(\bar{x}), f_2(\bar{x}), \dots, f_k(\bar{x})]^T \quad (1.9)$$

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

In other words, we wish to determine from among the set of all numbers which satisfy (1.7) and (1.8) the particular set  $x_1^*, x_2^*, \dots, x_k^*$  which yields the optimum values of all the objective functions.

The constraints given by (1.7) and (1.8) define the **feasible region**  $X$  and any point  $\bar{x}$  in  $X$  defines a **feasible solution**. The vector function  $\bar{f}(\bar{x})$  is a function which maps the set  $X$  in the set  $F$  which represents all possible values of the objective functions. The  $k$  components of the vector  $\bar{f}(\bar{x})$  represent the

non-commensurable criteria which must be considered. The constraints  $g_i(\bar{x})$  and  $h_i(\bar{x})$  represent the restriction imposed on the decision variables. The vector  $\bar{x}^*$  will be reserved to denote the optimal solutions (normally there will be more than one).

In multiobjective optimization problems, we may have to either

- Minimize all the objective functions
- Maximize all the objective functions
- Minimize some and maximize others

For simplicity reasons, normally all the functions are converted to a maximization or minimization form. For example, the following identity may be used to convert all the functions which are to be maximized into a form which allows their minimization:

$$\max f_i(\bar{x}) = -\min(-f_i(\bar{x})) \quad (1.10)$$

Similarly, the inequality constraints of the form

$$g_i(\bar{x}) \leq 0 \quad i = 1, 2, \dots, m \quad (1.11)$$

can be converted to (1.7) form by multiplying by  $-1$  and changing the sign of the inequality. Thus (1.11) is equivalent to

$$-g_i(\bar{x}) \geq 0 \quad i = 1, 2, \dots, m \quad (1.12)$$

A multiobjective optimization problem can be written in a shortened form as

$$\bar{f}(\bar{x}^*) = \underset{\bar{x} \in X}{opt} \bar{f}(\bar{x}) \quad (1.13)$$

where

$$\bar{f} : X \longrightarrow R^k \quad (1.14)$$

$$X = \{\bar{x} \in R^n \mid \bar{g}(\bar{x}) \geq 0, \bar{h}(\bar{x}) = 0\} \quad (1.15)$$

Here “opt” is used to indicate the optimum of the vector function. The problem is that the meaning of **optimum** is not well defined in this context, since we rarely have an  $\bar{x}^*$  such that for all  $i = 1, 2, \dots, k$

$$\bigwedge_{\bar{x} \in X} (f_i(\bar{x}^*) \leq f_i(\bar{x})) \quad (1.16)$$

If this was the case, then  $\bar{x}^*$  would be a desirable solution, but we normally never have a situation, like this, in which all the  $f_i(\bar{x})$  have a minimum in  $X$  at a common point  $x^*$ . An example of this ideal situation is shown in Figure 1.1 (taken from Osyczka [2]). However, since this situation rarely happens in real-world problems, then we have to establish a certain criteria to determine what would be considered an “optimal” solution. There are two main standard criteria for this, but before reviewing them, we need a few more definitions.

Let us assume that we find separately the minimum (or maximum) of all the objective functions. Assuming that they can be found, let

$$\bar{x}^{0(i)} = [x_1^{0(i)}, x_2^{0(i)}, \dots, x_n^{0(i)}]^T \quad (1.17)$$

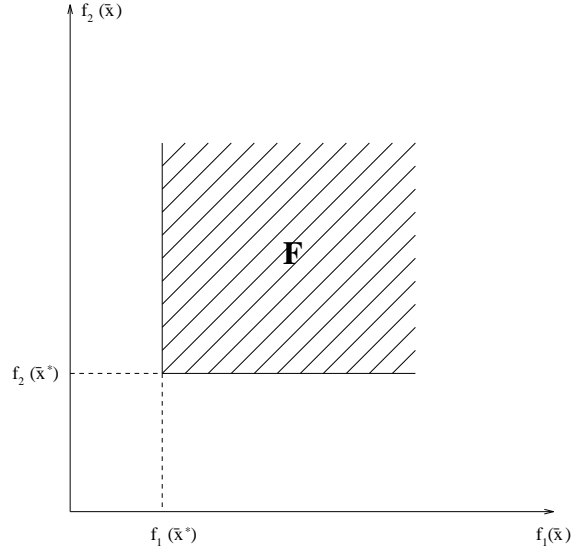


Figure 1.1: Ideal solution in which all our functions have their minimum at a common point.

be a vector of variables which optimizes (either minimizes or maximizes) the  $i$ th objective function  $f_i(x)$ . In other words, the vector  $\bar{x}^{0(i)} \in X$  is such that

$$f_i(\bar{x}^{0(i)}) = \underset{x \in X}{opt} f_i(\bar{x}) \quad (1.18)$$

In general, there will be a unified criteria with respect to “opt”. Most authors prefer to treat it as a minimum. In that case,  $f_i^0$  will denote the minimum value of the  $i$ th function. Hence, the vector  $\bar{f}^0 = [f_1^0, f_2^0, \dots, f_k^0]^T$  is ideal for a multiobjective optimization problem, and the point in  $R^n$  which determined this vector is the ideal (utopical) solution, and is called the **ideal vector**. Although this solution is generally not feasible, this is an important definition that will be used later.

Also, we have to define **convexity**. The set  $F$  is convex if for every  $\bar{a}^1$ ,  $\bar{a}^2 \in F$  and every  $\theta \in [0, 1]$

$$\bar{f}(\theta \bar{a}^1 + (1 - \theta) \bar{a}^2) \leq \theta \bar{f}(\bar{a}^1) + (1 - \theta) \bar{f}(\bar{a}^2) \quad (1.19)$$

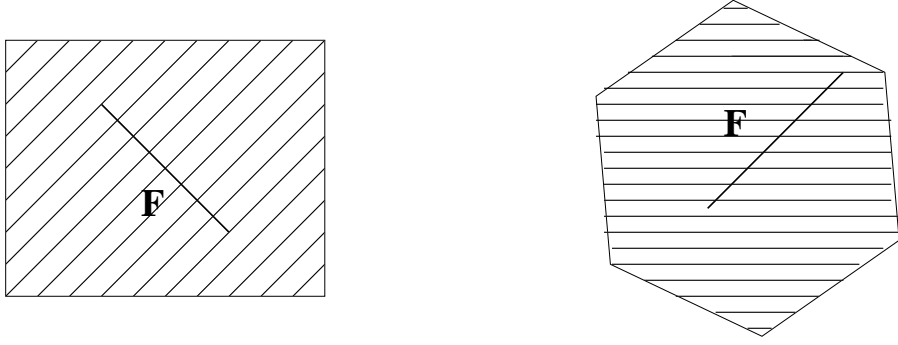


Figure 1.2: Two examples of convex sets.

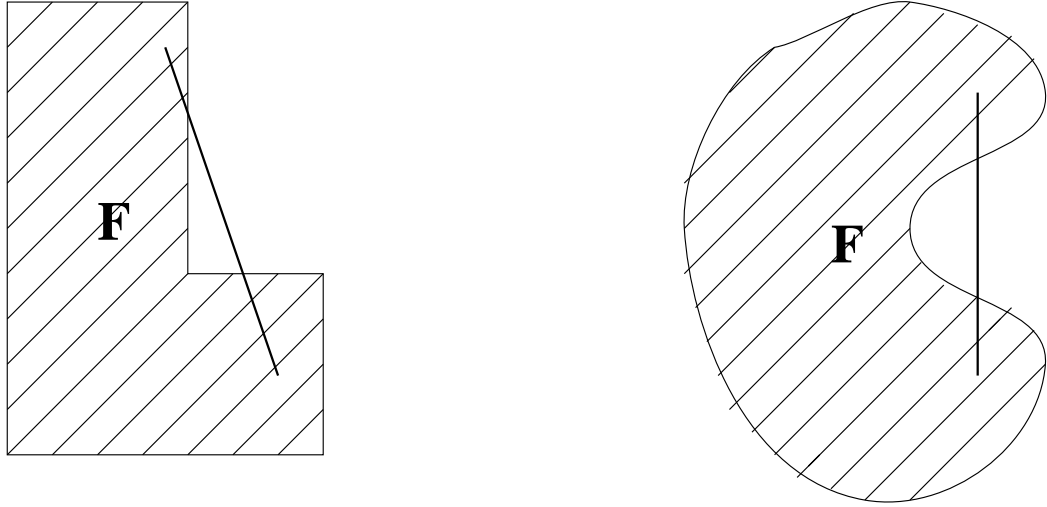


Figure 1.3: Two examples of non-convex sets.

In other words, the  $F$  is convex if for any points  $\bar{a}^1$  and  $\bar{a}^2$  in the set, the line segment joining these points is also in the set. So, for example, the sets shown in Figure 1.2 are convex, and the sets shown in Figure 1.3 are not.

We now need to define the  $t$ -**directional shadow** for the range  $F$ . This shadow is denoted by  $F^t$  and is defined as:

$$F^t = \{y \in R^k \mid y = \bar{f} + \alpha \bar{t}, \bar{f} \in F^p, \alpha \in R^1, \alpha \geq 0\} \quad (1.20)$$

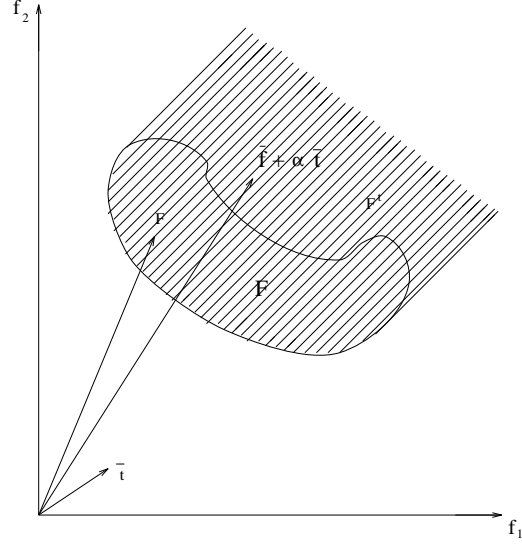


Figure 1.4: Graphical representation of the  $t$ -directional shadow for the range  $F$ .

A graphical representation of  $F^t$  is shown in Figure 1.4 (taken from Osyczka [2]).

From now on, we will call the multiobjective optimization problem convex if  $F^t$  is convex for all  $\bar{t} > 0$ . Some mathematical programming techniques that require the problem to be convex. The main issue is that there is no analytical method to classify a problem as being convex or non-convex.

Since, as we said before, there is no clear notion of optimum in multiobjective optimization, there are two main interpretations of this term:

- Pareto optimum
- Min-max optimum

The concept of **Pareto optimum** was formulated by Vilfredo Pareto in 1896 [3], and constitutes by itself the origin of research in this area. We say that a point  $\bar{x}^* \in X$  is **Pareto optimal** if for every  $\bar{x} \in X$  either,

$$\bigwedge_{i \in I} (f_i(\bar{x}) = f_i(\bar{x}^*)) \quad (1.21)$$

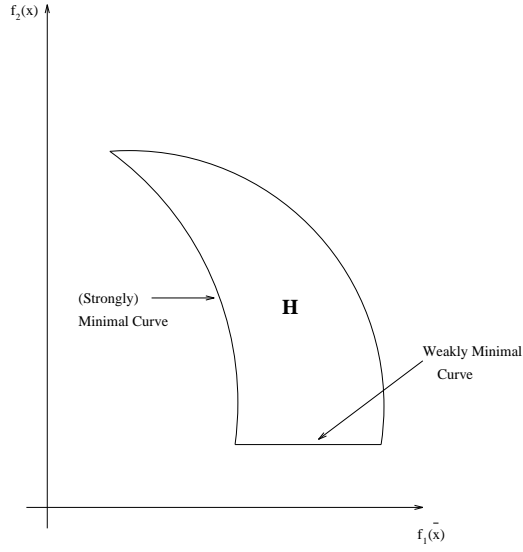


Figure 1.5: Weakly and strongly non-dominated curves on the biobjective case.

or, there is at least one  $i \in I$  such that

$$f_i(\bar{x}) > f_i(\bar{x}^*) \quad (1.22)$$

In words, this definition says that  $\bar{x}^*$  is Pareto optimal if there exists no feasible vector  $\bar{x}$  which would decrease some criterion without causing a simultaneous increase in at least one criterion. Unfortunately, the Pareto optimum almost always gives not a single solution, but a set of solutions called non-inferior or non-dominated solutions.

A point  $\bar{x}^* \in X$  is a **weakly non-dominated solution** if there is no  $\bar{x} \in X$  such that  $f_i(\bar{x}) < f_i(\bar{x}^*)$ , for  $i = 1, \dots, n$ .

A point  $\bar{x}^* \in X$  is a **strongly non-dominated solution** if there is no  $\bar{x} \in X$  such that  $f_i(\bar{x}) \leq f_i(\bar{x}^*)$ , for  $i = 1, \dots, n$  and for at least one value of  $i$ ,  $f(\bar{x}) < f(\bar{x}^*)$ .

Thus, if  $\bar{x}^*$  is strongly non-dominated, it is also weakly non-dominated, but the converse is not necessarily true. Non-dominated solutions for the biobjective

case can readily be represented graphically by passing into the objective function space  $\{f_1(\bar{x}), f_2(\bar{x})\}$ . To the locus of strongly non-dominated points corresponds the so-called **minimal curve**, and to the local of weakly non-dominated points, the **weakly minimal curve** [4]. These two curves are sketched in Figure 1.5 (taken from Duckstein [5]). We will use  $X^p$  to denote the set of noninferior or nondominated solutions, and  $F^p$  to denote the map of  $X^p$  in the space of objectives. The set  $X^p$  is, of course, determined from the set  $F^p$  which satisfies (1.21) and (1.22).

In engineering optimization, strongly non-dominated solutions are sought, and the qualificative “strongly” is generally omitted [5]. Let  $H$  be the set, called the pay-off set and shown in Figure 1.5, be defined by:

$$H = \{\bar{a} | \bar{a} = (a_i) \in R^n, \bar{x} \in X \text{ such that } a_i = f_i(\bar{x}) \text{ for every } i\} \quad (1.23)$$

Then, as proved by Szidarovszky and Duckstein [6], if  $H$  is non-empty closed and for every  $i$

$$\max \{a_i | (a_i) \in H\} < \infty, \quad (1.24)$$

then  $H$  has at least one strongly non-dominated solution. Thus, a large class of multiobjective optimization problems in engineering design may be expected to possess at least one non-dominated solution, and usually the problem is, as we mentioned before, that there is great number of possible solutions to choose from, and this may cause difficulties both in generating the solution set and in handling the results [7] [8].



The idea of stating the **min-max optimum** and applying it to multiobjective optimization problems, was taken from game theory, which deals with solving conflicting situations. The min-max approach to a linear model was proposed by Jutler [9] and Solich [10]. It has been further developed by Osyczka [11] [12], Rao [13] and Tseng and Lu [14].

The min-max optimum compares relative deviations from the separately attainable minima. Consider the  $i$ th objective function for which the relative deviation can be calculated from

$$z'_i(\bar{x}) = \frac{|f_i(\bar{x}) - f_i^0|}{|f_i^0|} \quad (1.25)$$

or from

$$z''_i(\bar{x}) = \frac{|f_i(\bar{x}) - f_i^0|}{|f_i(\bar{x})|} \quad (1.26)$$

It should be clear that for (1.25) and (1.26) we have to assume that for every  $i \in I$  and for every  $\bar{x} \in X$ ,  $f_i(\bar{x}) \neq 0$ .

If all the objective functions are going to be minimized, then equation (1.25) defines function relative increments, whereas if all of them are going to be maximized, it defines relative decrements. Equation (1.26) works conversely.

Let  $\bar{z}(\bar{x}) = [z_1(\bar{x}), \dots, z_i(\bar{x}), \dots, z_k(\bar{x})]^T$  be a vector of the relative increments which are defined in  $R^k$ . The components of the vector  $z(\bar{x})$  will be evaluated from the formula

$$\bigwedge_{i \in I} (z_i(\bar{x}) = \max \{z'_i(\bar{x}), z''_i(\bar{x})\}) \quad (1.27)$$

Now we define the min-max optimum as follows [2]:

A point  $\bar{x}^* \in X$  is min-max optimal, if for every  $\bar{x} \in X$  the following recurrence formula is satisfied:

Step 1:

$$v_1(\bar{x}^*) = \min_{\bar{x} \in X} \max_{i \in I} \{z_i(\bar{x})\} \quad (1.28)$$

and then  $I_1 = \{i_1\}$ , where  $i_1$  is the index for which the value of  $z_1(\bar{x})$  is maximal.

If there is a set of solutions  $X_1 \subset X$  which satisfies Step 1, then

Step 2:

$$v_2(\bar{x}^*) = \min_{\bar{x} \in X_1} \max_{i \in I, i \notin I_1} \{z_i(\bar{x})\} \quad (1.29)$$

and then  $I_2 = \{i_1, i_2\}$ , where  $i_2$  is the index for which the value of  $z_i(x)$  in this step is maximal.

If there is a set of solutions  $X_{r-1} \subset X$  which satisfies step  $r - 1$  then

Step r:

$$v_r(\bar{x}^*) = \min_{\bar{x} \in X_{r-1}} \max_{i \in I, i \notin I_{r-1}} \{z_i(\bar{x})\} \quad (1.30)$$

and then  $I_r = \{I_{r-1}, i_r\}$ , where  $i_r$  is the index for which the value of  $z_i(\bar{x})$  in the  $r$ th step is maximal.

If there is a set of solutions  $X_{k-1} \subset X$  which satisfies Step  $k - 1$ , then

Step k:

$$v_k(\bar{x}^*) = \min_{\bar{x} \in X_{k-1}} z_i(\bar{x}) \text{ for } i \in I \text{ and } i \notin I_{k-1} \quad (1.31)$$

where  $v_1(\bar{x}^*), \dots, v_k(\bar{x})$  is the set of optimal values of fractional deviations ordered non-increasingly.

This optimum can be described in words as follows. Knowing the extremes of the objective functions which can be obtained by solving the optimization problems for each criterion separately, the desirable solution is the one which gives the smallest values of the relative increments of all the objective functions.

The point  $\bar{x}^* \in X$  which satisfies the equations of Steps 1 and 2 may be called the best compromise solution considering all the criteria simultaneously and on equal terms of importance. It should be noticed that even when these equations look quite complicated, in many optimization models, only the first Step of this process will be necessary to determine the optimum.

In most cases, there will be several optimal solutions in the Pareto sense, and the designer will have to look to the values of the objective functions corresponding to  $F(X^P)$  in order to decide which value seems the most appropriate. This process in which a solution is accepted is called the **decision making** process.

It is easier to find the desirable solution from  $X^P$  if we know in advance the relative importance of each criteria, and in fact, some methods require this information. However, as in many cases we will not be able to provide this information because it will be incomplete or can not be expressed in a fully formalized way, then these methods are normally not used in practice.

The minima in the Pareto sense are going to be in the boundary of the design region, or in the locus of the tangent points of the objective functions. Figure 1.6 (taken from Hernández [15]) shows these boundaries shaded. In general, it is not easy to find an analytical expression of the line or surface that contains these points, and the normal procedure is to compute the points  $X^P$  and their

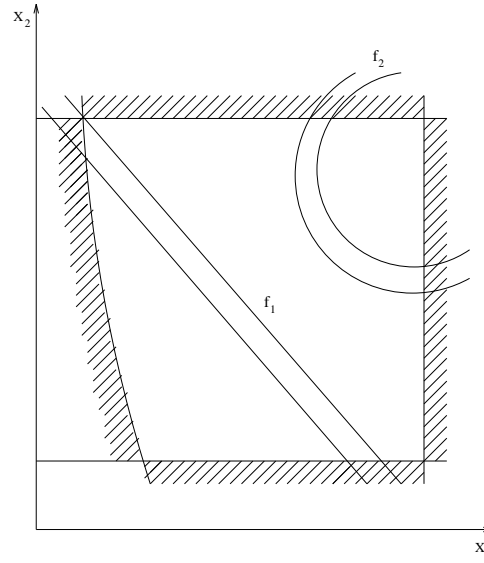


Figure 1.6: An example of a problem with two variables and two objective functions. The pareto optimal solutions are indicated by the shaded boundaries of the design region.

corresponding  $F(X^p)$ . When we have a sufficient amount of these, we may proceed to take the final decision.

In general, we can say that if all the criteria are equally important in a problem, then the optimum in the min-max sense may give us a desirable solution. In all other cases a solution from the set of optimal solutions in the Pareto sense should be chosen.

The following example, taken from Hernández [15] should help to better understand the concepts covered in this chapter:

Consider the plane truss shown in Figure 1.7 (taken from Hernández [15]). The following data are assumed:  $P = 2.1t$ ,  $\sigma_i = 2.1 \text{ t/cm}^2$  ( $i=1,2$  and  $3$ ),  $E = 2100 \text{ t/cm}^2$ ,  $x_1 \leq 1.414 \text{ cm}^2$ , and  $x_2 \leq 2.828 \text{ cm}^2$ . Where  $P$  is the load applied to the structure,  $\sigma$  the maximum allowed stress of the material,  $E$  the modulus of elasticity of the material, and  $x_1$  and  $x_2$  the cross-sectional areas of the truss.

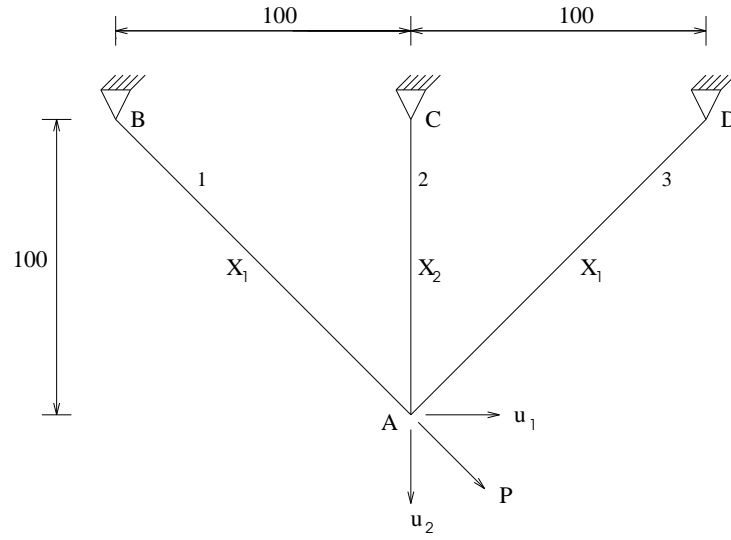


Figure 1.7: A three-bar plane truss used to illustrate the basic concepts covered in this chapter.

We want to minimize the following objective functions:

$$V = 100 \left( 2\sqrt{2}x_1 + x_2 \right) \quad (1.32)$$

$$u_2 = \frac{100P}{E \left( x_1 + \sqrt{2}x_2 \right)} \quad (1.33)$$

subject to:

$$g_1 \equiv \sigma_1 = \frac{x_2 + \sqrt{2} x_1}{2x_1x_2 + \sqrt{2} x_1^2} P \leq 2.1 \quad (1.34)$$

$$g_2 \equiv \sigma_2 = \frac{\sqrt{2} x_1}{2x_1x_2 + \sqrt{2} x_1^2} P \leq 2.1 \quad (1.35)$$

$$g_3 \equiv \sigma_3 = \frac{x_2}{2x_1x_2 + \sqrt{2} x_1^2} \leq 2.1 \quad (1.36)$$

$$0 \leq x_1 \leq 1.414 \quad (1.37)$$

$$0 \leq x_2 \leq 2.828 \quad (1.38)$$

where  $V$  stands for the total volume of the structure, and  $u_2$  refers to the vertical displacement of node A. Similarly  $u_1$  in Figure 1.7 refers to the horizontal displacement of node A. So, what we want is to minimize the total volume of the truss, at the same time that we want minimum vertical displacement of Node A. The only constraints are those imposed by the stress and allowed cross-sectional areas of the structural elements. As we can see, what we really are going to optimize is a function of the cross-sectional areas, so the **decision variables** are  $x_1$  and  $x_2$ . Equations (1.34), (1.35), (1.36), (1.37) and (1.38) are all **inequality constraints**. We do not have **equality constraints** in this case, but we could be required, for example, to satisfy  $x_1 + x_2 = 4.5$ . This would be an equality constraint. Equations (1.32) and (1.33) are the **objective functions**.

Each one of the constraints  $g_i(\bar{x}) \leq 0$  determines a portion of the search space where we can find a solution. The intersection of all of them determines the **design region**. A point  $\bar{x}$  that belongs to this region is a **valid design**, and it will be an **invalid design** if it is outside. Points inside the region are called **unconstrained designs**, and are called **constrained designs** when they are on the line  $g_i(\bar{x}) = 0$ , or in the intersection of several of them. Constraints in which  $g_i(\bar{x}) = 0$  are called **active**, and are called **passive** otherwise. We may have constraints that are placed outside the design region, in which case they will not be relevant to the optimization process, and we can ignore them. In this example,  $g_2(\bar{x}) \leq 0$  only has one point in the design region:  $(1, 0)$ . However, this fact is not always easy to identify in problems with many variables.

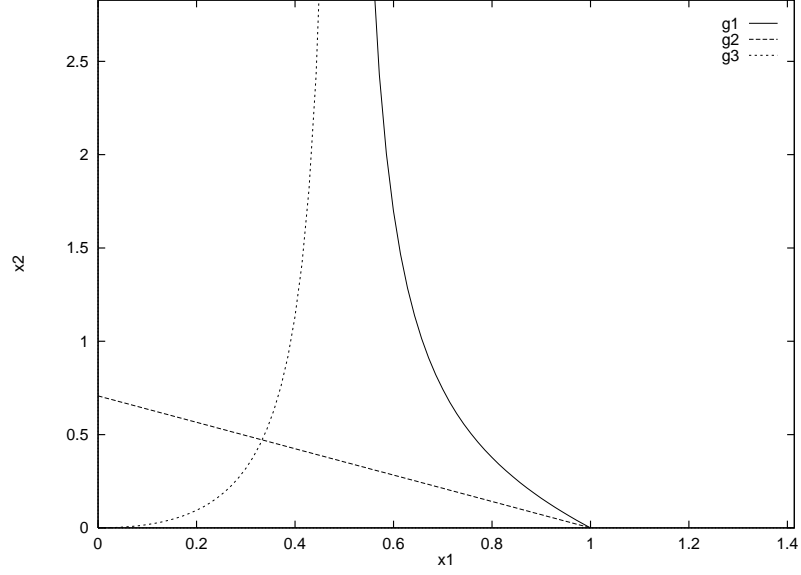


Figure 1.8: Euclidean space of the decision variables for the three-bar truss of Figure 1.7. The curve  $g_1$  limits the feasible region.

Figure 1.8 shows the Euclidean space of the decision variables. The shaded area is called the **feasible region** or **design region**, and it is normally denoted by  $X$ . Any point  $\bar{x} \in X$  defines a feasible solution—i.e., a solution that satisfies all the constraints. Figure 1.8 was generated by rewriting equations (1.34), (1.35) and (1.36) into the following forms:

$$g_1 \equiv x_2 = \frac{\sqrt{2} x_1^2 - \sqrt{2} x_1}{1 - 2x_1} \quad (1.39)$$

$$g_2 \equiv x_2 = \frac{\sqrt{2} (1 - x_1)}{2} \quad (1.40)$$

$$g_3 \equiv x_2 = \frac{\sqrt{2} x_1^2}{1 - 2x_1} \quad (1.41)$$

which was possible by making  $g_i = 0$ ,  $i = 1, 2, 3$ .

To be able to define the Pareto minima, we need to find the geometrical loci that form the tangent points of the objective functions [15]. We will first

determine the coordinates of point  $x_e$ , which corresponds to the minimum volume. According to equation (1.32), the slope of equal volume equals  $(-2\sqrt{2})$ . We can find the coordinates of  $x_e$  by drawing a straight line having  $(-2\sqrt{2})$  as a slope and meeting the curve  $g_1$  without entering the feasible region. This can be achieved by making the tangent to  $g_1$  equal to  $(-2\sqrt{2})$ , that is

$$\frac{2\sqrt{2} x_1 - 2\sqrt{2} x_1^2 - \sqrt{2}}{(1 - 2x_1)^2} = -2\sqrt{2} \quad (1.42)$$

Then,  $x_1 = 0.788675$  and  $x_2 = 0.408249$ .

Now we need to find the coordinates of point  $x_b$ , which corresponds to the minimum vertical displacement. By observing equation (1.33), we can easily infer that the minimum will occur at the maximum values of  $x_1$  and  $x_2$ , which are  $\sqrt{2}$  and  $2\sqrt{2}$ , respectively.

We also need to define three additional points:  $x_a, x_c$  and  $x_d$ . The first corresponds to the point where the straight horizontal line passing by  $x_b$  cuts  $g_1$ , the second to the point where a vertical line passing by  $x_b$  cuts the abscissas axis, and the last corresponds to the point where  $g_1$  cuts the abscissas axis. Then,  $x_a$  has the same vertical coordinate than  $x_b$  ( $2\sqrt{2}$ ). To find its horizontal coordinate, we need to make equation (1.39) equals to  $2\sqrt{2}$ :

$$x_2 = \frac{\sqrt{2} x_1^2 - \sqrt{2} x_1}{1 - 2x_1} = 2\sqrt{2} \quad (1.43)$$

from which

$$x_1^2 + 3x_1 - 2 = 0 \quad (1.44)$$

is derived, and finally, we have:  $x_1 = 0.561553$ .



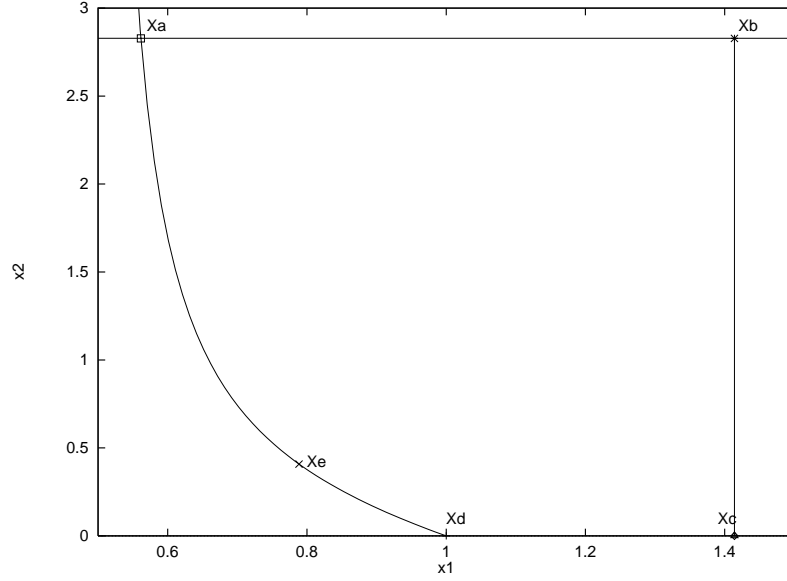


Figure 1.9: Design region for the three-bar plane truss of Figure 1.7.

The coordinates of  $x_c$  are trivial to find, since  $x_2 = 0$  and  $x_1 = \sqrt{2}$ . Finally, to find the horizontal coordinate of  $x_d$ , we use:

$$x_2 = \frac{\sqrt{2} x_1^2 - \sqrt{2} x_1}{1 - 2x_1} = 0 \quad (1.45)$$

from which  $x_1 = 1$ .

The Pareto minima are in the boundary region defined by  $x_a$ ,  $x_b$  and  $x_e$ . The feasible region is bounded by  $x_a$ ,  $x_b$ ,  $x_c$  and  $x_d$ . These boundaries are shown in Figure 1.9.

The boundary of the feasible region in the objective functions space is found by eliminating  $x_1$  and  $x_2$  between equations (1.32) and (1.33), and each one of the constraints. The equations found in each case are the following:

1. For the line  $x_a, x_b$ , we have:

$$x_2 = 2 \sqrt{2} \quad (1.46)$$

By using equation (1.33) and (1.46), we have:

$$u_2 = \frac{0.1}{x_1 + 4} \quad (1.47)$$

Therefore:  $x_1 = \frac{0.1}{u_2} - 4$ .

Using this result and equation (1.32), we have:

$$V = 200 \sqrt{2} \left[ \frac{0.1 - 3u_2}{u_2} \right] \quad (1.48)$$

2. For the line  $x_b, x_c$ , we have:

$$x_1 = \sqrt{2} \quad (1.49)$$

By using equation (1.33) and (1.49), we have:

$$u_2 = \frac{0.1}{\sqrt{2}(1 + x_2)} \quad (1.50)$$

Therefore:  $x_2 = \frac{0.1}{\sqrt{2} u_2} - 1$ .

Using this result and equation (1.32), we have:

$$V = \frac{300 \sqrt{2} u_2 + 10}{\sqrt{2} u_2} \quad (1.51)$$

3. For the line  $x_c, x_d$ , we have:

$$x_2 = 0 \quad (1.52)$$

By using equation (1.33) and (1.52), we have:

$$u_2 = \frac{0.1}{x_1} \quad (1.53)$$

Therefore:  $x_1 = \frac{0.1}{u_2}$ .

Using this result and equation (1.32), we have:

$$V = \frac{20 \sqrt{2}}{u_2} \quad (1.54)$$

4. For the line  $x_a, x_d$ , we have:

$$x_2 = \frac{\sqrt{2} x_1^2 - \sqrt{2} x_1}{1 - 2x_1} \quad (1.55)$$

By using equation (1.33) and (1.55), we have:

$$u_2 = 0.2 - \frac{0.1}{x_1} \quad (1.56)$$

Therefore:  $x_1 = \frac{0.1}{0.2 - u_2}$ .

Using this result and equation (1.32), we have:

$$V = \frac{10 \sqrt{2} u_2 + \sqrt{2}}{0.2u_2 - u_2^2} \quad (1.57)$$

Figure 1.10 shows the solution region. The values corresponding to the Pareto minima are between  $F(x_b)$  and  $F(x_e)$ . Looking at this graph, the decision maker can think about the evolution of each objective function, and then decide on the most appropriate combination. For example, for this case, the point  $F(x_f)$  with coordinates (0.05, 282.8427125) seems a very good choice, since it provides greater rigidity with a slight increment of volume. The design variables corresponding to this point are:  $x_1 = \frac{2}{3}$  and  $x_2 = 0.9428090416$ .

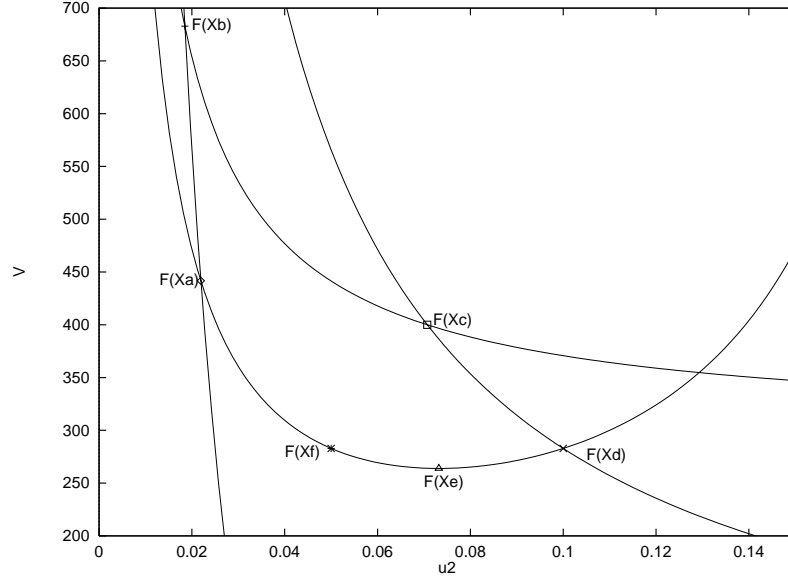


Figure 1.10: The solution region for the three-bar plane truss of Figure 1.7 is delimited by points  $F(x_a)$ ,  $F(x_b)$ ,  $F(x_c)$ ,  $F(x_d)$  and  $F(x_e)$ . The point  $F(x_f)$  corresponds to the solution adopted.

If we want to use the min-max method to solve this problem, a common approach would be to normalize the objective functions within the range  $[0, 1]$  by using the expression [15]:

$$\bar{f}_k = \frac{f_k - \min f_k}{\max f_k - \min f_k} \quad k = 1, \dots, n \quad (1.58)$$

This means that we need to know the minima and maxima of all the objective functions. In this case, that can be easily determined by looking at Figure 1.10. From there, we have:

$$\min V = 263.8958434 \quad (1.59)$$

$$\max V = 682.8427125 \quad (1.60)$$

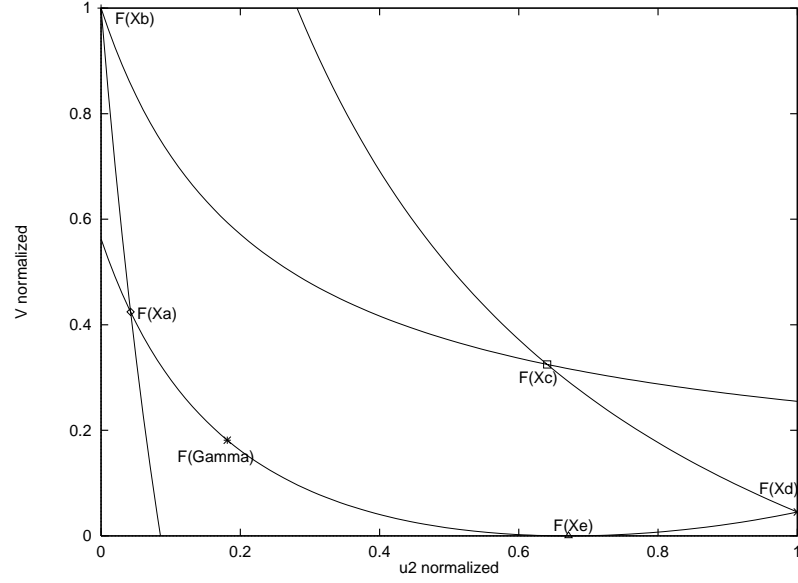


Figure 1.11: The normalized solution region for the three-bar plane truss of Figure 1.7 is delimited by points  $F(x_a)$ ,  $F(x_b)$ ,  $F(x_c)$ ,  $F(x_d)$  and  $F(x_e)$ . The point  $F(\text{Gamma})$  corresponds to the solution adopted.

Then,  $\bar{V}$  can be defined as:

$$\bar{V} = \frac{200 \sqrt{2} x_1 + 100x_2 - 263.8958434}{418.9468691} \quad (1.61)$$

Similarly, for the vertical displacement, we have:

$$\text{Min } u_2 = 0.01846990313 \quad (1.62)$$

$$\text{Max } u_2 = 0.1 \quad (1.63)$$

And  $\bar{u}_2$  can be defined as:

$$\bar{u}_2 = \frac{\frac{0.1}{x_1 + \sqrt{2} x_2} - 0.01846990313}{0.08153009687} \quad (1.64)$$

The (normalized) solution region is shown in Figure 1.11.

The formulation of the min-max problem is:

$$\text{Min } \gamma \tag{1.65}$$

subject to:

$$\frac{200 \sqrt{2} x_1 + 100x_2 - 263.8958434}{418.9468691} \leq \gamma \tag{1.66}$$

$$\frac{\frac{0.1}{x_1 + \sqrt{2} x_2} - 0.01846990313}{0.08153009687} \leq \gamma \tag{1.67}$$

and constraints (1.34) to (1.38).

The solution is:  $\gamma = \bar{V} = \bar{u}_2 = 0.1813243496$ .

This corresponds to the decision variables:  $x_1 = 0.599712$  and  $x_2 = 1.702365795$ . The corresponding values of the objective functions at this point are:  $V = 339.8607483$  and  $u_2 = 0.03325329492$ .

# Chapter 2

## Mathematical Programming

### Techniques

Multiobjective optimization theory is not as recent as we might think, since Kuhn and Tucker [16] and Koopmans [17] must be credited with its discovery in 1951. However, multiobjective optimization theory remained relatively undeveloped from 1951 until the 1960's when multiobjective public investment problems became more common and "trade-off" became a favorite word of managers, planners, and decision makers. So, this area arose in a natural fashion in mathematical economics, and many techniques were developed by systems analysts and decision theorists for private and public sector problems, by control theorists for engineering (guidance and design) problems, and by water resource economists and systems analysts for water resource planning problems. Good reviews of the mathematical programming techniques for multiobjective optimization can be found in Terry [18], Kapur [19], Roy [20], Loucks [21], Cohon and Marks [22], Wierzbicki [23], Hwang and Masud [8], Hwang et al. [24], Ignizio [25], Osyczka and Koski [26], Stadler [27], Starr and Zeleny [28], Lieberman [29], Evans [30] and Fishburn [31]. These papers provide a very rich bibliography on the subject.

I will try to cover the main methods found in the literature, giving a brief description of each one, and the main references where more specific information may be found. The initial arrangement of these techniques was taken from Duckstein [5].

## 2.1 The Sequential Optimization Method

This method can be applied only if a preference order can be assigned on the  $k$  objectives. Assume that the first objective function is the most important for the decision maker, and the objective functions have been ranked according to the priority list  $f_1, f_2, \dots, f_k$ . One first solves the problem

$$\begin{aligned} \min f_1(\bar{x}) \\ \text{subject to } \bar{x} \in X \end{aligned} \tag{2.1}$$

If  $\alpha_1$  denotes the optimal value of  $f_1$ , then in the second step one solves the problem:

$$\begin{aligned} \min f_2(\bar{x}) \\ \text{subject to } f_1(\bar{x}) = \alpha_1 \\ \bar{x} \in X \end{aligned} \tag{2.2}$$

If  $\alpha_2$  denotes the optimal value of  $f_2$ , then the third step consists of solving the problem:

$$\begin{aligned} \min f_3(\bar{x}) \\ \text{subject to } f_i(\bar{x}) = \alpha_i \quad (i = 1, 2) \\ \bar{x} \in X \end{aligned} \tag{2.3}$$

and so on.



In general, one solves the following problem at the  $v$ th step:

$$\begin{aligned}
 & \min f_v(\bar{x}) \\
 & \text{subject to } f_i(\bar{x}) = \alpha_i \quad (i = 1, \dots, v-1) \\
 & \bar{x} \in X
 \end{aligned} \tag{2.4}$$

If problem (2.4) is unbounded, then we say that there is no solution. If part of problem (2.4) has a unique optimum or  $v = k$ , then this solution is selected as a pre-emptive optimal solution; otherwise, one proceeds to the  $(v + 1)$ st step. In other words, if problem (2.2) has a unique solution  $\bar{x}^*$ , which would be the case in many convex programming problems, that solution is also the one selected for the multiobjective problem. This technique, therefore, may not be appropriate for engineering optimization problems where unique solutions are frequent and where a real trade-off between conflicting objectives is desired. Readers interested in this method should refer to Benayoun et al. [32] for more information.

## 2.2 The Weighting Objectives Method

This method consists of adding all the objective functions together using different weighting coefficients for each one of them. This means that our multi-objective optimization problem is transformed into a scalar optimization problem:

$$\min \sum_{i=1}^k w_i f_i(\bar{x}) \tag{2.5}$$

where  $w_i \geq 0$  are the weighting coefficients representing the relative importance of the objectives. It is usually assumed that

$$\sum_{i=1}^k w_i = 1 \tag{2.6}$$

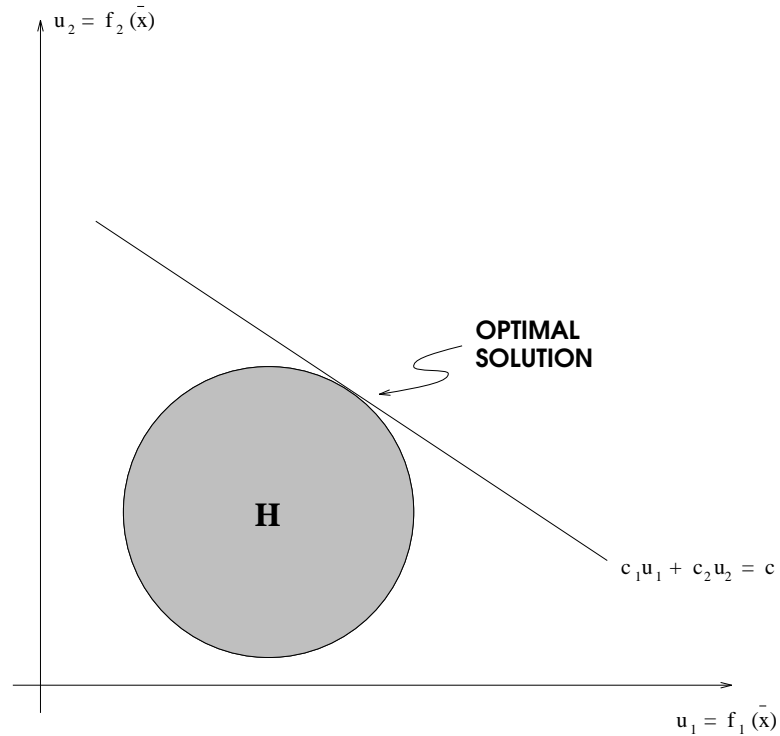


Figure 2.1: The weighting objectives method for a maximizing problem.

Since the results of solving an optimization model using (2.5) can vary significantly as the weighting coefficients change, and since very little is usually known about how to choose these coefficients, a necessary approach is to solve the same problem for many different values of  $w_i$ . But in this case, the designer is still, of course, confronted with the decision of having to choose the most appropriate solution based on his intuition.

Note that the weighting coefficients do not reflect proportionally the relative importance of the objectives, but are only factors which, when varied, locate points in the Pareto set. For the numerical methods that can be used to seek the minimum of (2.5), this location depends not only on  $w_i$  values, but also on the units in which the functions are expressed. The solution of the single objective programming problem (2.5) (in a maximizing case) is shown in Figure 2.1 (taken

from Duckstein [5]), where  $H$  denotes the feasible pay-off set and the linear function  $\sum_{i=1}^k c_i u_i$  is maximized subject to  $\bar{u}$  being in the feasible set  $H$ . Note that  $\max y = \min (-y)$ .

If we want  $w_i$  to reflect closely the importance of the objectives, all functions should be expressed in units of approximately the same numerical values. We can also transform (2.5) to the form:

$$\min \sum_{i=1}^k w_i f_i(\bar{x}) c_i \quad (2.7)$$

where  $c_i$  are constant multipliers.

The best results are usually obtained if  $c_i = 1/f_i^0$ . In this case, the vector function is normalized to the form  $\bar{f}(\bar{x}) = [\bar{f}_1(\bar{x}), \bar{f}_2(\bar{x}), \dots, \bar{f}_k(\bar{x})]^T$ , where  $\bar{f}_i(\bar{x}) = f_i(\bar{x})/f_i^0$ .

A condition  $f_i^0 \neq 0$  is assumed and if it is not satisfied, which rarely happens [2], the value of  $c_i$  must be chosen by the decision maker.

Szidarovszky and Duckstein [6] proved that:

- If  $\bar{x}^*$  is an optimal solution of (2.5), then  $\bar{x}^*$  is weakly non-dominated; if  $\bar{x}^*$  is a unique solution, then it is also strongly non-dominated.
- If  $w_i > 0$  for all  $i = 1, \dots, k$ , then any optimal solution  $\bar{x}$  of (2.5) is strongly non-dominated.

The weighting objectives method was the first technique developed for the generation of non-inferior solutions for multiobjective optimization. This is an obvious consequence of the fact that it was implied by Kuhn and Tucker [16]. This method is very efficient computationally speaking, and can be applied to generate a strongly non-dominated solution to be used as an initial solution in other techniques. If it is relatively easy to obtain the feasible pay-off set  $H$ , then

this method can be quite appropriate. For more information on this method, refer to Cohon [33].

## 2.3 The $\varepsilon$ -constraint Method

This method is based on minimization of one (the most preferred or primary) objective function, and considering the other objectives as constraints bound by some allowable levels  $\varepsilon_i$ . Hence, a single objective minimization is carried out for the most relevant objective function  $f_1$  subject to additional constraints on the other objective functions. The levels  $\varepsilon_i$  are then altered to generate the entire Pareto optima set. The method may be formulated as follows:

- (1) Find the minimum of the  $r$ th objective function, i.e., find  $\bar{x}^*$  such that

$$f_r(\bar{x}^*) = \min_{\bar{x} \in X} f_r(\bar{x}) \quad (2.8)$$

subject to additional constraints of the form

$$f_i(\bar{x}) \leq \varepsilon_i \text{ for } i = 1, 2, \dots, k \text{ and } i \neq r \quad (2.9)$$

where  $\varepsilon_i$  are assumed values of the objective functions which we wish not to exceed.

- (2) Repeat (1) for different values of  $\varepsilon_i$ . The information derived from a well chosen set of  $\varepsilon_i$  can be useful in making the decision. The search is stopped when the decision maker finds a satisfactory solution.

It may be necessary to repeat the above procedure for different indices  $r$ .

To get adequate  $\varepsilon_i$  values, single-objective optimizations are normally carried out for each objective function in turn by using mathematical programming techniques. For each objective function  $f_i$  ( $i = 1, 2, \dots, k$ ), there is an optimal

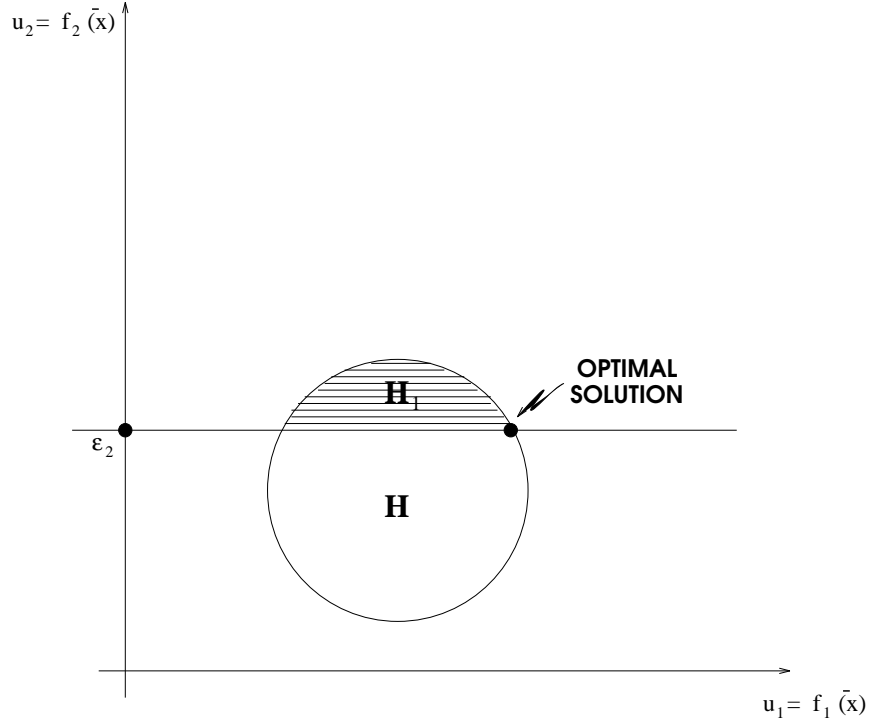


Figure 2.2: The  $\varepsilon$ -constraint method for a maximizing problem.

design vector  $\bar{x}_i^*$  for which  $f_i(\bar{x}_i^*)$  is a minimum. Let  $f_i(\bar{x}_i^*)$  be the lower bound on  $\varepsilon_i$ , i.e.

$$\varepsilon_i \geq f_i(\bar{x}_i^*) \quad i = 1, 2, \dots, r-1, r+1, \dots, k \quad (2.10)$$

and  $f_i(\bar{x}_r^*)$  be the upper bound on  $\varepsilon_i$ , i.e.

$$\varepsilon_i \leq f_i(\bar{x}_r^*) \quad i = 1, 2, \dots, r-1, r+1, \dots, k \quad (2.11)$$

When the bounds  $\varepsilon_i$  are too low, there is no solution and at least one of these bounds must be relaxed.

Figure 2.2 (taken from Duckstein [5]) illustrates the  $\varepsilon$ -constraint method for a maximizing problem where  $H$  is the payoff set of the original problem, restricted to the shadowed area  $H_1$  by the further constraint  $f_2(\bar{x}) \geq \varepsilon_2$  (we are

maximizing), and the objective function  $f_1$  is maximized subject to the assumption that  $\bar{x}$  belongs to  $H_1$ . Thus, the most important objective (in this case,  $f_1$ ) has been optimized, and the others, as I said before, are handled as additional constraints.

Szidarovszky and Duckstein [6] showed that the  $\varepsilon$ -constraint method usually leads to weakly non-dominated solutions; however, if the optimal solution is unique, then it is strongly non-dominated.

This method, also known as **trade-off method**, because of its main concept of trading a value of one objective function for a value of another function, is further explained in Osyczka [2], Lounis and Cohn [34], Carmichael [35], and Hwang et al. [24].

## 2.4 Global Criterion Method

In this method, we try to minimize a function which defines a global criterion which is a measure of how close the decision maker can get to the ideal vector  $\bar{f}^0$ . The most common form of this function is [2]

$$f(\bar{x}) = \sum_{i=1}^k \left( \frac{f_i^0 - f_i(\bar{x})}{f_i^0} \right)^p \quad (2.12)$$

For this formula Boychuk and Ovchinnikov [36] have suggested  $p = 1$ , and Salukvadze [37] has suggested  $p = 2$ , but other values of  $p$  can also be used. Obviously, the results will differ greatly depending on the value of  $p$  chosen. Thus, the selection of the best  $p$  is an issue in this method, and it could also be the case that any  $p$  could produce an unacceptable solution.

Another possible measure of ‘closeness to the ideal solution’ is a family of  $L_p$ -metrics defined as follows

$$L_p(f) = \left[ \sum_{i=1}^k |f_i^0 - f_i(x)|^p \right]^{1/p}, \quad 1 \leq p \leq \infty \quad (2.13)$$

In general, relative deviations of the form

$$\frac{f_i^0 - f_i(x)}{f_i^0} \quad (2.14)$$

are preferred over absolute deviations, because they have a substantive meaning in any context. The relevant  $L_p$  metrics are

$$L_p(f) = \left[ \sum_{i=1}^k \left| \frac{f_i^0 - f_i(\bar{x})}{f_i^0} \right|^p \right]^{1/p}, \quad 1 \leq p \leq \infty \quad (2.15)$$

The value of  $p$  indicates the type of distance: for  $p = 1$ , all deviations from  $f_i^*$  are taken into account in direct proportion to their magnitudes, which corresponds to ‘group utility’ [38] [39]. For  $2 \leq p < \infty$ , the larger deviations carry greater weight in  $L_p$ ; for  $p = \infty$ , the largest deviation is the only one taken into consideration, which leads to a purely ‘individual utility’ (min-max criterion), in which all weighted deviations are equal.

Koski [40] [7] has suggested  $L_p$ -metrics with a normalized vector objective function of the form

$$\bar{f}_i(x) = \frac{f_i(\bar{x}) - \min_{x \in X} f_i(\bar{x})}{\max_{x \in X} f_i(\bar{x}) - \min_{x \in X} f_i(\bar{x})} \quad (2.16)$$

In this case, the values of every normalized function are limited to the range  $[0,1]$ .

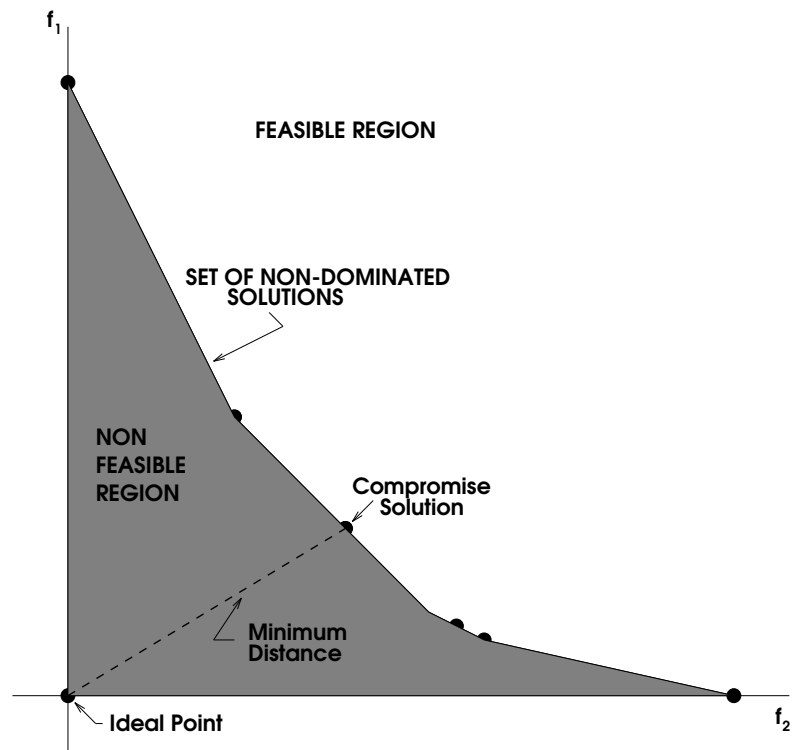


Figure 2.3: Sketch of a compromise solution. The basic idea is to take the point which is closest, by some distance measure, to the ideal point, which in this case is the origin.



Using the global criterion method one non-inferior solution is obtained. If certain parameters  $w_i$  are used as weights for the criteria, a required set of non-inferior solutions can be found. Duckstein [5] calls this method **Compromise Programming**, and his  $L_p$ -metrics is

$$L_p(\bar{x}) = \left[ \sum_{i=1}^k w_i^p \left| \frac{f_i(\bar{x}) - f_i^0}{f_{i \max} - f_i^0} \right|^p \right]^{1/p} \quad (2.17)$$

where  $w_i$  are the weights,  $f_{i \max}$  is the worst value obtainable for criterion  $i$ ;  $f_i(\bar{x})$  is the result of implementing decision  $\bar{x}$  with respect to the  $i$ th criterion. Figure 2.3 (taken from Duckstein and Opricovic [39]) shows an sketch of a compromise solution.

The ‘displaced ideal’ technique [41] which proceeds to define an ideal point, a solution point, another ideal point, etc. is an extension of compromise programming.

Another variation of this technique is the method suggested by Wierzbicki [42] [43] in which the global function has a form such that it penalizes the deviations from the so-called reference objective. Any reasonable or desirable point in the space of objectives chosen by the decision maker can be considered as the reference objective.

Let  $\bar{f}^r = [f_1^r, f_2^r, \dots, f_k^r]^T$  be a vector which defines this point. Then the function which is minimized has the form

$$P(\bar{x}, \bar{f}^r) = - \sum_{i=1}^k i = 1^k (f_i(\bar{x}) - f_i^r)^2 + \varrho \sum_{i=1}^k (\max(0, f_i(\bar{x}) - f_i^r))^2 \quad (2.18)$$

where  $\varrho > 0$  is a penalty coefficient which in this method can be chosen as constant.

Minimizing (2.18) for the assumed point  $\bar{f}^r$  we obtain the non-inferior solution, which is close to this point. If for different points  $\bar{f}^r$  the procedure is carried out, some representation of non-inferior solutions can be found.

More information on this method can be found in Osyczka [2] and Zeleny [44] [45].

## 2.5 Goal Programming

Charnes and Cooper [46] and Ijiri [47] are credited with the development of the goal programming method for a linear model, and have played a key role in applying it to industrial problems. Although initially the method was developed for multiobjective optimization, its subsequent use justifies the credit given to Charnes and Cooper for their work in this area.

In this method, the decision maker has to assign targets or goals that he wishes to achieve for each objective. These values are incorporated into the problem as additional constraints. The objective function will then try to minimize the absolute deviations from the targets to the objectives. The simplest form of this method may be formulated as follows [5]:

$$\min \sum_{i=1}^k |f_i(\bar{x}) - T_i|, \quad \text{subject to } \bar{x} \in X \quad (2.19)$$

where  $T_i$  denotes the target or goal set by the decision maker for the  $i$ th objective function  $f_i(\bar{x})$ , and  $X$  represents the feasible region. The criterion, then, is to minimize the sum of the absolute values of the differences between target values and actually achieved values. A more general formulation of the goal programming objective function is a weighted sum of the  $p$ th power of the

deviation  $|f_i(\bar{x}) - T_i|$  [48]. Such a formulation has been called **generalized goal programming** [49] [50].

Looking again to equation (2.19), the objective function is non-linear and the simplex method can be applied only after transforming this equation into a linear form, thus reducing goal programming to a special type of linear programming. In this transformation [46] new variables  $d_i^+$  and  $d_i^-$  are defined such that:

$$d_i^+ = \frac{1}{2}\{|f_i(\bar{x}) - T_i| + [f_i(\bar{x} - T_i)]\}, \quad (2.20)$$

$$d_i^- = \frac{1}{2}\{|f_i(\bar{x}) - T_i| - [f_i(\bar{x} - T_i)]\}, \quad (2.21)$$

Adding and subtracting these equations, the following equivalent linear formulation may be found:

$$\min Z_0 = \sum_{i=1}^k (d_i^+ + d_i^-), \quad (2.22)$$

subject to

$$\begin{aligned} \bar{x} &\in X \\ f_i(\bar{x}) - d_i^+ + d_i^- &= T_i \\ d_i^+, d_i^- &\geq 0, \quad i = 1, \dots, k \end{aligned} \quad (2.23)$$

Since we can not have both under- and overachievements of the goal simultaneously, then at least one of the deviational variables must be zero. In other words:

$$d_i^+ \cdot d_i^- = 0 \quad (2.24)$$

Fortunately, this constraint is automatically fulfilled by the simplex method because the objective function will drive either  $d_i^+$  or  $d_i^-$  or both variables simultaneously to zero for all  $i$ .

Sometimes it may be desirable to express preference for over- or underachievement of a goal. Thus, it may be more desirable to overachieve a targeted reliability figure than to underachieve it. To express preference for deviations, the decision maker can assign relative weights  $w_i^+$  and  $w_i^-$  to positive and negative deviations, respectively, for each target  $T_i$ . If a minimization problem is considered, choosing the  $w_i^+$  to be larger than  $w_i^-$  would be expressing preference for underachievement of a goal.

In addition, Goal programming provides the flexibility to deal with cases with conflicting multiple goals. Essentially, the goals may be ranked in order of importance to the problem solver. That is, a priority factor,  $p_i$  ( $i = 1, \dots, k$ ) is assigned to the deviational variables associated with the goals. These factors  $p_i$  are conceptually different from weights, as it is explained, for example, in Goicoechea et al. [51]. The resulting optimization model becomes

$$\min S_0 = \sum_{i=1}^k p_i (w_i^+ d_i^+ + w_i^- d_i^-), \quad (2.25)$$

subject to

$$\begin{aligned} \bar{x} &\in X \\ f_i(\bar{x}) - d_i^+ + d_i^- &= T_i \\ d_i^+, d_i^- &\geq 0, \quad i = 1, \dots, k \end{aligned} \quad (2.26)$$

Note that this technique will yield a dominated solution if the goal point is chosen in the feasible domain [5]. This technique has been applied in a few structural optimization applications [52] [14], because of the inherently non-linear

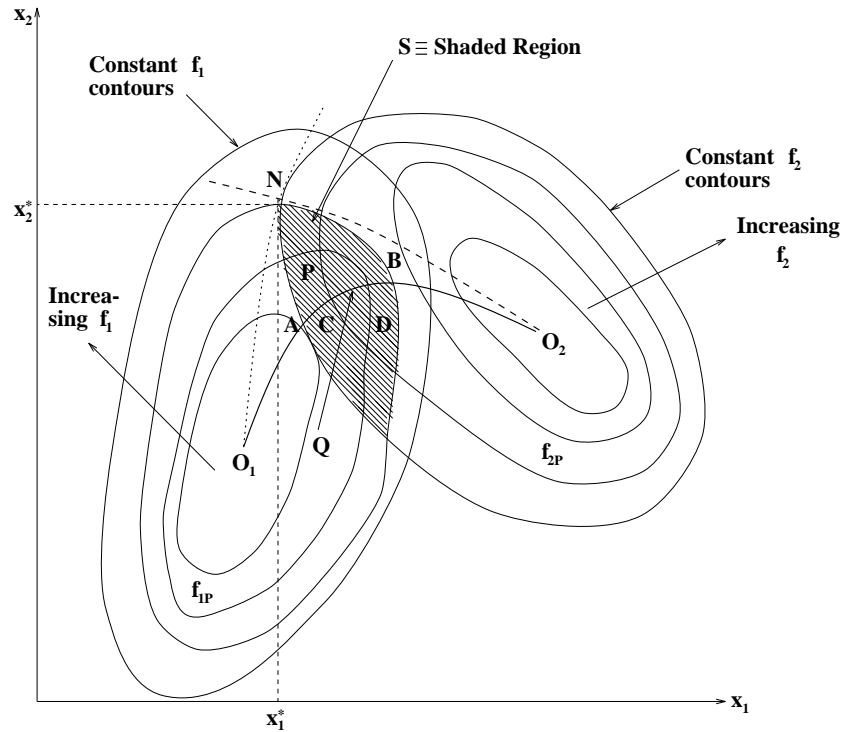


Figure 2.4: Example of cooperative and non-cooperative game solutions.

nature of structural problems. However, it can be very useful in those cases in which a linear or piecewise-linear approximation can be made, because of the availability of excellent computer programs, and the possibility of eliminating dominated goal points easily. On the other hand, in non-linear cases, other techniques guaranteeing a non-dominated solution may be preferable. Also, we can say that this technique is very efficient, computationally speaking, but it has as another drawback the fact that when using priorities, these have to be devised by the decision maker without prior knowledge of the alternatives.

More information on this method can be found in Charnes [53], Lee and Jaaskelainen [54], Lee [55] and Ignizio [49].

## 2.6 Game Theory Approach

We can analyze this technique with reference to a two objective, two design variable optimization problem whose graphical representation is shown in Figure 2.4 (taken from Rao [56]). Let  $f_1(x_1, x_2)$  and  $f_2(x_1, x_2)$  represent two scalar objectives and  $x_1$  and  $x_2$  two scalar design variables. It is assumed that one player is associated with each objective. The first player wants to select a design variable  $x_1$  which will minimize his objective function  $f_1$ , and similarly the second player seeks a variable  $x_2$  which will minimize his objective function  $f_2$ . If  $f_1$  and  $f_2$  are continuous, then the contours of constant values of  $f_1$  and  $f_2$  appear as shown in Figure 2.4. The dotted lines passing through  $O_1$  and  $O_2$  represent the loci of rational (minimizing) choices for the first and second player for a fixed value of  $x_2$  and  $x_1$ , respectively. The intersection of these two lines, if it exists, is a candidate for the two objective minimization problem, assuming that the players do not cooperate with each other (**non-cooperative game**). In Figure 2.4, the point  $N(x_1^*, x_2^*)$  represents such a point. This point, known as a Nash equilibrium solution, represents a stable equilibrium condition in the sense that no player can deviate unilaterally from this point for further improvement of his own criterion [57].

This point has the characteristic that

$$f_1(x_1^*, x_2^*) \leq f_1(x_1, x_2^*) \quad (2.27)$$

and

$$f_2(x_1^*, x_2^*) \leq f_2(x_1^*, x_2) \quad (2.28)$$

where  $x_1$  may be to the left or right of  $x_1^*$  in Equation (2.27) and  $x_2$  may lie above or below  $x_2^*$  in Equation (2.28).

Extension of the idea to a  $k$ -player non-cooperative game gives the mathematical definition of a **Nash equilibrium solution** as

$$\left. \begin{aligned} f_1(x_1^*, x_2^*, \dots, x_k^*) &\leq f_1(x_1, x_2^*, \dots, x_k^*) \\ f_2(x_1^*, x_2^*, \dots, x_k^*) &\leq f_2(x_1^*, x_2, \dots, x_k^*) \\ &\vdots \\ f_k(x_1^*, x_2^*, \dots, x_k^*) &\leq f_k(x_1^*, x_2^*, \dots, x_k) \end{aligned} \right\} \quad (2.29)$$

The problem becomes more interesting when there is more than one Nash equilibrium point. In that case, since the values of  $f_1$  and  $f_2$  are different Nash equilibrium points, any player can have the advantage of declaring his/her move first thereby forcing the other players to play at the equilibrium point of his/her own choice.

In a **cooperative game**, the two players agree to cooperate with each other and hence any point in the shaded region  $S$  of Figure 2.4 will provide both of them with a better solution than their respective Nash equilibrium solutions [58]. Since the region  $S$  does not provide a unique solution, the concept of Pareto optimality may be introduced to reduce the number of solutions. It can be seen that all points in the region  $S$  can be eliminated except those on the continuous line  $O_1ACQDBO_2$  which represents the loci of tangent points between the contours of  $f_1$  and  $f_2$ .

Every point on this line has the property that it is not dominated by any other point in its neighborhood, i.e.

$$f_1(Q) \leq f_1(P) \quad (2.30)$$

and

$$f_2(Q) \leq f_2(P), \quad (2.31)$$

where  $Q$  is a point lying on the line  $O_1O_2$  and  $P$  is a neighboring point. Thus all points of  $S$  that do not lie on the line  $O_1O_2$  need not be considered during cooperative play. The set of all points lying on  $AB$  is the Pareto-optimal set, and can be denoted as  $S_p$ .

The cooperative game theory approach of solving the multiobjective optimization problem can be stated as follows:

The  $k$  players are assumed to correspond to the  $k$  objectives; one for each objective. While playing the game, each player will try to improve his/her own conditions (i.e., to decrease the value of his/her own objective function). The players will start bargaining from their respective reference (starting) values and put a joint effort in maximizing a subjective criterion (super-criterion) formed by themselves. It is assumed that each player has analyzed his/her own criterion before starting the game to find the maximum possible benefit he/she can obtain. This will also help him/her in guaranteeing against the worst value. This analysis is necessary since each player should know the extreme conditions of his/her own and others so that none of them begins bargaining from a reference value which is unrealistic (i.e., unacceptable to the other players).



The extreme values for each player are determined as follows:

At any starting feasible design vector  $\bar{x}_s$ , the objective function values are found and positive constant multipliers  $m_1, m_2, \dots, m_k$  are chosen such that

$$m_1 f_1(\bar{x}_s) = m_2 f_2(\bar{x}_s) = \dots = m_k f_k(\bar{x}_s) = M, \quad (2.32)$$

where  $M$  is a constant. By redefining the objective functions as

$$F_i(\bar{x}) = m_i f_i(\bar{x}); \quad i = 1, 2, \dots, k, \quad (2.33)$$

one can notice that any design vector which minimizes the function  $f_i(\bar{x})$  will also minimize the function  $F_i(\bar{x})$ . This scaling is done to make all the objective functions numerically equal at a particular design  $\bar{x}_s$ . Hereafter, it will be assumed that the  $k$  players correspond to the  $k$  scaled objective functions given by Equation (2.33).

Starting from the design vector  $\bar{x}_s$ , each objective function  $F_i(\bar{x})$ ;  $i = 1, 2, \dots, k$  is minimized subject to the constraints

$$g_i(\bar{x}) \leq 0, \quad i = 1, 2, \dots, m \quad (2.34)$$

Then a matrix  $[P]$  is constructed as

$$[P] = \begin{bmatrix} F_1(\bar{x}_1^*) & F_2(\bar{x}_1^*) & F_k(\bar{x}_1^*) \\ F_1(\bar{x}_2^*) & F_2(\bar{x}_2^*) & F_k(\bar{x}_2^*) \\ \vdots & & \\ F_1(\bar{x}_k^*) & F_2(\bar{x}_k^*) & F_k(\bar{x}_k^*) \end{bmatrix} \quad (2.35)$$

It can be seen that the diagonal elements in the matrix  $[P]$  are the minima in their respective columns.

Defining  $F_{iu}$  as

$$F_{iu} = \max F_i(\bar{x}_j^*); \quad j = 1, 2, \dots, k; \quad i = 1, 2, \dots, k, \quad (2.36)$$

a rectangular matrix  $[R]$  is constructed as

$$[R] = \begin{bmatrix} F_1(\bar{x}_1^*) & F_{1u} \\ F_2(\bar{x}_2^*) & F_{2u} \\ \vdots & \vdots \\ F_i(\bar{x}_i^*) & F_{iu} \\ \vdots & \vdots \\ F_k(\bar{x}_k^*) & F_{ku} \end{bmatrix} \quad (2.37)$$

This matrix gives the extreme values of all the players. For example, the  $i$ th row, which corresponds to the  $i$ th player, indicates that during the cooperative play, he/she should not expect a value for his/her objective better than  $F_i(\bar{x}_i^*)$  but is guaranteed that his/her objective will never be worse than  $F_{iu}$ .

Assuming that all the players start negotiation by taking their worst values as reference values, a supercriterion  $S$  can be constructed as:

$$S = \prod_{i=1}^k \{F_{iu} - F_i(\bar{x}^*)\}, \quad (2.38)$$

where  $\bar{x}_c^*$ , a Pareto-optimal solution, minimizes a combined objective function  $F_c(\bar{c}, \bar{x})$  defined by

$$F_c(\bar{c}, \bar{x}) = \sum_i^k c_i F_i(\bar{x}) \quad (2.39)$$

subject to the constraints of Equation (2.34), and  $c_i$  satisfies the conditions

$$c_i \geq 0, \quad i = 1, 2, \dots, k \quad (2.40)$$

and

$$\sum_{i=1}^k c_i = 1 \quad (2.41)$$

From Equation (2.38) it can be seen that all the players will be interested in maximizing the criterion  $S$ . As  $\bar{x}_c^*$  is implicitly independent on the values of  $c_i$ , the problem is now to determine the optimum values of  $c_i$  for which  $S$  is maximum.

The procedure of obtaining the optimum vector

$$\bar{c}^* = \left\{ \begin{array}{c} c_1^* \\ \vdots \\ c_k^* \end{array} \right\} \quad (2.42)$$

begins by assuming any vector  $\bar{c}$  and improving it in the subsequent iterations by moving along the steepest ascent directions of  $S$  through appropriate step lengths.

For more information on this method, refer to Rao [56] [52], Szidarovszky et al. [59], and Gershon et al. [60].

## 2.7 Metagames and Hypergames

Metagame analysis, or the analysis of options, is a game-theoretic technique which was originally suggested to analyze real-world political problems. When using this kind of analysis, a conflict is considered as a game in which the major decision-making parties, or players, have countable options. A possible selection

	preferred by A	particular outcome	not preferred by A	infeasible
<b>Player A</b>				
Option A1	1 1	0	0 0 1 0 1	10110101
Option A2	0 1	1	0 0 0 1 1	01100011
<b>Player B</b>				
Option B1	0 0	0	0 1 1 1 1	00001111
Option B2	1 1	1	0 0 0 0 0	00011111

Figure 2.5: Preferences for player A.

of options for one particular player is referred to as a **strategy**, and the situation where each player chooses a strategy is called an **outcome**.

Figure 2.5 shows a game in tabular form where there are two players and each player has two options. A number one in front of an option indicates the inclusion of the option in a player's strategy, while a zero indicates that the option is not taken. A strategy for a player is formed by a combination of ones and zeros against all of the players' options in Figure 2.5 (taken from Hipel and Fraser [61]). The farthest left outcome in the same figure is the situation where player A incorporates option A1 into his/her strategy while player B uses option B2. This outcome is written horizontally as [1 0 0 1].

The particular outcome is examined from the point of view of a particular player who is player A in Figure 2.5 (taken from Hipel and Fraser [61]). There are 3 categories into which outcomes may fall with respect to the preferences of the particular player. They may be preferred to the particular outcome, they may be not preferred to the particular outcome, or they may be infeasible (the set of infeasible outcomes is often common to all players). Within these categories, outcomes can be more economically displayed by combining the outcomes in a

	preferred by A	particular outcome	not preferred by A	infeasible
<b>Player A</b>				
Option A1	1	0	1 - -	- 1 1 -
Option A2	-	1	0 - -	1 - 0 -
<b>Player B</b>				
Option B1	0	0	- 1 -	0 0 - 1
Option B2	1	1	- - 0	0 0 1 1

unilateral improvement

inescapable sanction

Figure 2.6: Generalized preferences for player A.

specified manner to obtain columns containing dashes. The dashes represent both a '0' and a '1' for the option opposite to the dash. Thus, a column containing  $n$  dashes represents  $2^n$  outcomes. Columns containing dashes can be generated from Figure 2.5 by employing a technique called **generalization** [62]. Infeasible outcomes are generalized separately with both the preferred and nonpreferred outcomes to enhance the interpretation of the tabular form of the game. The outcomes from Figure 2.5 can be generalized to obtain Figure 2.6.

For this particular example, player A can unilaterally move to a preferred position if the options of the other players remain the same. This is called a unilateral improvement and is indicated by the arrow pointing to the left at the bottom of Figure 2.6 (taken from Hipel and Fraser [61]). Because there is a unilateral improvement, the particular outcome is not rational for the particular player.

However, in the example given, player B can select the strategy [1 0] so that no matter what A does, he will be in a less preferred position relative to the

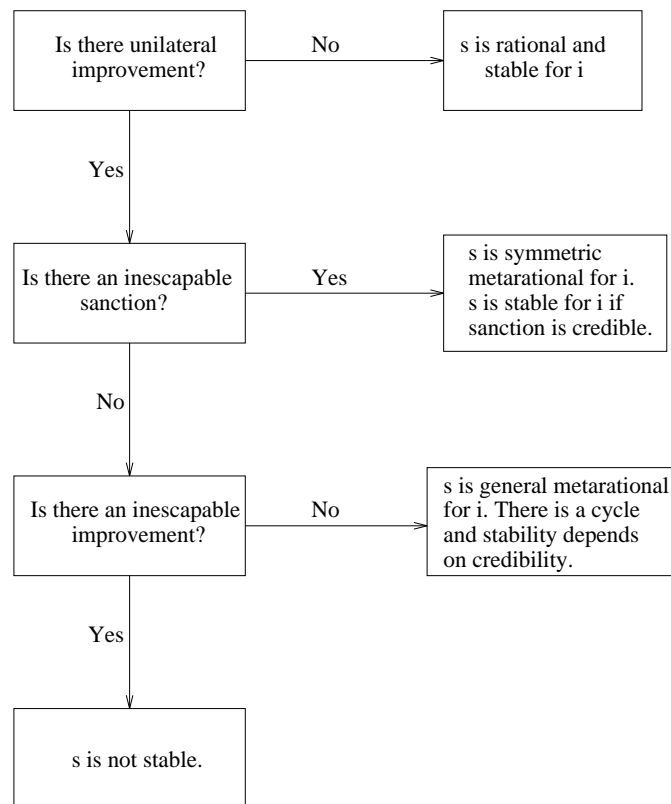


Figure 2.7: Stability analysis of a particular outcome  $s$  for a particular player  $i$ .

particular outcome. This is called an **inescapable sanction**, and if credible to A, he will be deterred from moving away from the particular outcome. This is indicated by the arrow pointing to the right in Figure 2.6. The particular outcome is then called **symmetric metarational** and thus is stable for A.

If there were a column on the preferred side for player A such that dashes were entered against all options of player B (in general all the other players), then A could make an inescapable improvement. A unilateral improvement which is not inescapable and does not have an inescapable sanction is **general metarational**, and stability depends on credibility. Neither of these situations are shown in Figure 2.6.

The analysis of a particular outcome for a particular player is outlined in Figure 2.7. Starting in the upper left hand box, the particular outcome is checked for the characteristics listed. When the bottom or a right-hand box is encountered, the stability of the particular outcome for the particular player is determined. If a particular outcome is stable for all players, it is a possible resolution or equilibrium to the conflict.

More information on metagame theory and analysis may be found in Howard [63] and Hipel et al. [62] [64]. Fraser and Hipel developed a conflict analysis method [65] [66] [67] that is an extension of metagame analysis.

A **Hypergame** is what classical game theorists refer to as a game of incomplete information. In a hypergame, one or more of the players may possess a mistaken view of the actual conflict where misunderstandings can be built upon others' misperceptions in order to form different levels of perception [68]. The players in a hypergame may have a false understanding of the preferences of the other players, have an incorrect comprehension of the options available to the

other players, not be aware of all the players in the game, or have any combination of the foregoing faulty interpretations. Indeed, the definition of a hypergame should also include the situation where a given player is not certain about his/her own preferences and options. This event could arise when a player consists of a set of individuals or organizations for which there is not unanimous agreement among the group members about the group's preferences and options.

A hypergame constitutes a hierarchy of games, whereby each participant's idea about what is happening is properly modeled, since the approach models individual games according to the way each player interprets the dispute. The level of a hypergame depends on expectation and equals the highest order of expectation involved. If all the players are playing the same game, the conflict is a simple game or a hypergame of level zero. If the players are playing different games and no one knows that, it is a hypergame of level one. In a second-level hypergame, at least one of the players knows that they are playing different games. Therefore, this player is trying to perceive what the other players' games are. This can be interpreted as the players playing different first-level hypergames. A third-level hypergame has to be employed if at least one of the players knows that they are playing different first-level hypergames. Thus, he tries to perceive the other players' first-level hypergames which forms his/her second-level perceptual game. The more sophisticated the players are, the higher the level of the hypergame is.

The hypergame structure allows the combination of different forms of games. The individual games are connected by a set of mappings which are functions transferring other players' viewpoints into a given player's perceptual games. The idea is that each game may be interpreted as the expression of the particular idea of a certain player of what is happening.



More information on hypergames may be found in Takahashi et al. [68], Hipel et al. [67], Okada et al. [69] and Wang et al. [70].

## 2.8 Multiattribute Utility Theory

Von Neumann and Morgenstern [71] developed an axiomatic utility theory to measure individual or group preferences. Utility theory assumes that an individual can choose among the alternatives available to him in such a manner that the satisfaction derived from his choice is as large as possible. This, of course, implies the individual is aware of his alternatives and is capable of evaluating them. Moreover, relative to a vector of objectives it is assumed all information pertaining to the various levels of the objectives can be captured by an individual's **utility function**. In effect, an individual's utility function is a formal, mathematical representation of his preference structure. Multiattribute utility functions, which may be assessed as first proposed in Keeney [72] and Raiffa [73], and then in Berger [74] and Keeney and Raiffa [75], integrate the objective functions into the preference structure. The highest degree of utility with respect to all the objectives is obtained by maximizing the utility function.

Oppenheimer [76] distinguishes two approaches to utility maximization: the global and the local approaches.

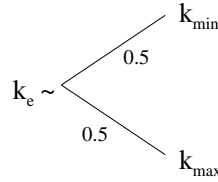
The **global approach** [77] means the above expected utility maximization, and 'may force the decision maker to fit a function not truly representing' the preference function. Nevertheless, the global approach is taken in most multiattribute utility models.

In the **local approach** [78], the above-mentioned problem of locking the decision maker into a given risk attitude is avoided by using a sequence of local linear approximations to the utility function. To each step pertains a trial solution

representing an improvement over its predecessor, so that eventually, the sequence reaches its optimum.

The main drawback of this approach is that the decision maker has to spend a lot of time building single-attribute utility functions, and then, he/she has to make sure that the ‘corner utilities’ are assessed; the latter make it possible to combine the single attribute utilities  $u_i(x_i)$  into one function  $u(\bar{x})$ .

To illustrate the assessment task, let four attributes,  $x_1, x_2, x_3$ , and  $x_4$ , be, respectively, the weight  $W$ , probabilities of failure  $(1 - r) = p_f$ , cost  $k$ , and deflection  $\Delta$  of a structure. The first task is to assess the function  $u_i(x_i)$ ,  $i = 1, 2, 3, 4$ ; this can be best done by means of lotteries of the type:



In words, given a lottery in which maximum cost  $k_{max}$  and minimum cost  $k_{min}$  may be obtained with equal probability 0.5 (for example), which value  $k_e$  would the decision maker accept as a ‘certainty equivalent’? Furthermore, if the axioms of Von Neuman and Morgenstern are satisfied, it can be proved [71] that a utility function  $u(\cdot)$  exists, leading to the equation:

$$u(k_e) = 0.5u(k_{max}) + 0.5u(k_{min}) \quad (2.43)$$

Utility functions are defined within a positive linear transformation and one usually sets  $u(k_{max}) = 0$  and  $u(k_{min}) = 1$  so that  $u(k_e) = 0.5$ . This procedure is continued in the intervals  $(k_{max}, k_e)$ ,  $(k_e, k_{min})$ , and overlapping intervals, until a satisfactory piecewise-linear approximation of the utility function is obtained.

If the attributes  $W$ ,  $1 - r$ ,  $k$  and  $\Delta$  are mutually utility independent, the function  $u(\bar{x})$  is given by [75]:

$$1 + ku(\bar{x}) = \prod_{i=1}^k [1 + kk_i u_i(x_i)] \quad (2.44)$$

Verification of the utility independence hypothesis, assessment of the  $k_i$  ( $i = 1, 2, 3, 4$ ), and consistency check require a further series of lotteries. However, even when a lot of effort is required to construct  $u(\bar{x})$ , it may be worth it in large and costly engineering systems [5].

For more information on this method, refer to Gershon et al. [60] and Krzysztofowicz and Duckstein [79].

## 2.9 Surrogate Worth Trade-Off

This method, proposed by Haimes and Hall [80], is a variant of the **trade-off** method in which objective trade-offs are used as the information carrier and the decision maker responds by expressing his/her degree of preference over the prescribed trade-offs by assigning numerical values to each surrogate worth function. These functions are used to construct a single objective problem. First, the set of strictly non-dominated or efficient solutions is generated, say by the  $\varepsilon$ -constraint method. Then a search along the efficient boundary is performed using a surrogate worth function. Note the difference between this method, which stays on the Pareto-optimum boundary, and compromise programming or game theory, in which the Pareto-optimum set is approached, respectively, from the infeasible and the feasible regions.

The trade-off function for any two objectives evaluated at a given efficient solution  $\bar{x}$  is:

$$T_{ij}(\bar{x}) = \frac{\partial f_i(\bar{x})}{\partial f_j(\bar{x})} \quad (2.45)$$

As we can see from Equation (2.45), this method can only be applied when all the objective functions are differentiable.

The surrogate worth function,  $W_{ij}$ ,  $i \neq j$ ,  $j = 1, 2, \dots, n$ , is defined as a function of the desirability of the trade-off  $\lambda_{ij}$  on a scale. For example, if we use a scale ranging from  $-10$  to  $+10$ , a  $(-10)$  would indicate that  $\lambda_{ij}$  marginal units of objective  $i$  are worth very much less than one marginal unit of objective  $j$ , a  $(+10)$  means the opposite, and a zero indicates an even trade-off. The best solution is found when all surrogate worth functions are equal to zero. A complete description of this technique can be found in Haimes et al. [48], and an abbreviated version, in Goicoechea et al. [51].

The main advantage of this technique resides in its sound theoretical basis and it is possible to find good set of published applications [81] [82] [83] [84]. On the other hand, computational requirements are non-trivial, and much input is required from the decision maker.

In general, we can say that the trade-off methods have two main disadvantages [2]:

- They cannot be used to solve non-convex problems, and
- they allow a satisfactory solution to be found only in a certain region of Pareto optimal solutions, but do not provide a general outlook on the possible range of objectives, and thus the final decision is influenced by the starting point chosen.

## 2.10 ELECTRE I and II

These two techniques are applicable to problems that have a discrete predefined set of alternatives in which some of the evaluation criteria are non-quantifiable, i.e., the criteria can only be ranked ordinally or, with additional information, on a ratio or interval scale.

**ELECTRE I** (elimination and (et) choice translating algorithm) was developed by Benayoun et al. [32], was improved by Roy [20] and it has been applied, for example, to water-related problems [85] [39] [86]. The idea is to choose those systems which are preferred for at least a plurality of the criteria and yet do not cause an unacceptable level of discontent for any one criterion. This methodology leans on three concepts: concordance, discordance, and threshold values.

The **concordance** between any two systems  $i$  and  $j$  is a weighted measure of the number of criteria for which action  $i$  is preferred to action  $j$  (denoted  $i \succ j$ ) or for which action  $i$  is equal to action  $j$  (denoted  $i \sim j$ ) and is given as:

$$C(i, j) = \frac{\sum_{k \in A(i, j)} w(k)}{\sum_k w(k)}, \quad (2.46)$$

where  $w(k)$  is the weight of criterion  $k$ ,  $k = 1, \dots, K$ , and  $A(i, j) = \{k | i \succeq j\}$ . The weights, which are given by the decision maker, reflect his/her preferences. Concordance may be considered as the weighted percentage of criteria for which one action is preferred to another. Note that, by construction,  $0 \leq C(i, j) \leq 1$ .

Determination of the **discordance** between  $i$  and  $j$  requires that an interval scale common to each criterion be defined. The scale is used to compare the discomfort caused between the ‘worst’ and the ‘best’ criterion value for each pair of alternatives. A range may be chosen where the ‘best’ rating would be assigned

the highest value of the range and the ‘worst’ rating would receive the lowest value of the range. Each criterion, however, can have a different range to reflect the ‘leeway’ available for that criterion [5]. The problem of applying a ratio scale to an ordinal criterion presents theoretical difficulties which are fully addressed in Zeleny [45] and Rietveld [87]. Essentially, evaluations of the type (a, b, c, d) may be assigned in an analogous way in which grades are assigned to students. The **discordance index** is defined as:

$$D(i, j) = \frac{\max_{k=1, K} (Z(j, k) - Z(i, k))}{R^*}, \quad (2.47)$$

where  $Z(j, k)$  is the evaluation of alternative  $j$  with respect to criterion  $k$ , and  $R^*$  is the largest of the  $K$  criterion scales. Again, by construction,  $0 \leq D(i, j) \leq 1$ .

To synthesize both, the concordance and discordance matrices, **threshold values** ( $p, q$ ) between zero and one, are defined by the problem solver. Using a geometric representation, the preference relationships define a transitive and complete graph ( $G$ ) for each criterion, in which nodes are alternatives and arcs are directed as the preference sign  $\succ$ . In the case of  $i \sim j$ , one arc is drawn from  $i$  to  $j$  and another from  $j$  to  $i$ . The arc set  $A$  of the composite graph ( $\Gamma$ ) which synthesizes both concordance and discordance relationships, is given by:

$$a(i, j) \in A \Leftrightarrow (C(i, j) > p) \cap (D(i, j) < q) \quad (2.48)$$

Figure 2.8 (taken from Goicoechea et al [51]) shows an example of the type of graph that ELECTRE I uses. In choosing the value of  $p$ , the problem solver specifies how much ‘concordance’ is wanted:  $p = 1$  corresponds to full concordance, which means that  $i$  should be preferred or equivalent to  $j$  in terms

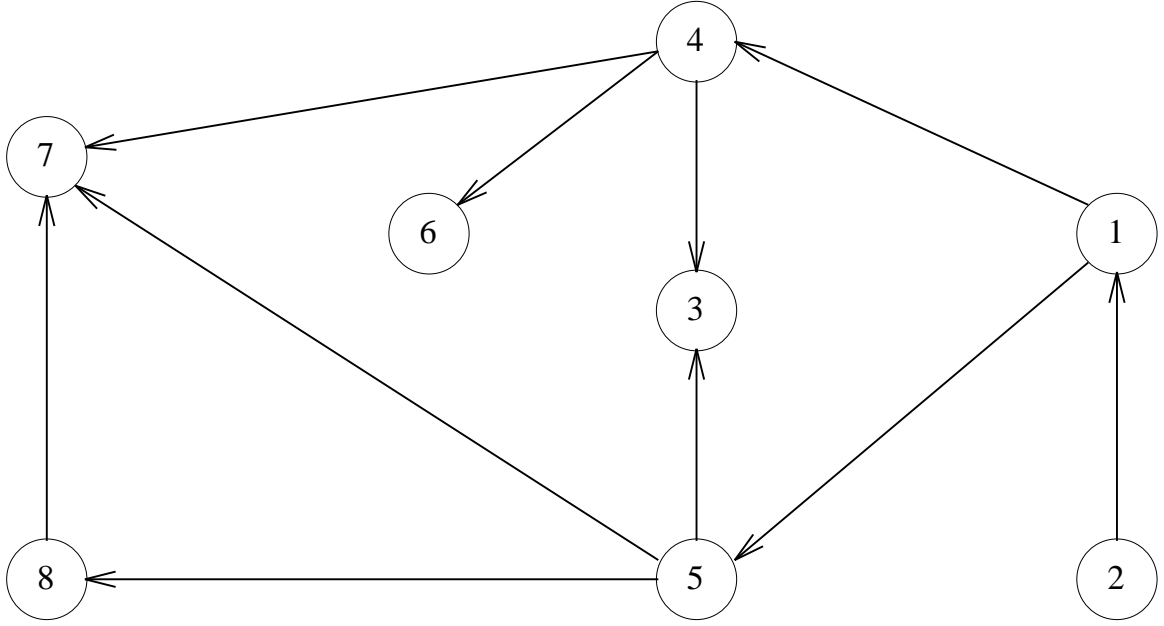


Figure 2.8: Example of an ELECTRE graph. Each node corresponds to a non-dominated alternative. The arrows indicate preferences. Therefore we can say that alternative 1 is preferred to alternative 5, alternative 4 is preferred to alternative 6, etc.

of all criteria. By choosing  $q$ , the amount of tolerable ‘discordance’ is specified:  $q = 0$  means no discordance. It is possible that some choices of  $p$  and  $q$  may eliminate all alternative systems. If this is the case, the values of  $p$  and/or  $q$  must be restated. It is also possible for cycles to occur in the composite graph ( $\Gamma$ ) of ELECTRE I. In such cases the nodes along the cycle are collapsed into one new node, which is equivalent to assigning the same ranking to those systems.

The preference graph ( $\Gamma$ ) of ELECTRE I thus yields a partial ordering of the alternative systems. On the other hand, **ELECTRE II** [20] [86], may be used to obtain a complete ordering, as in Duckstein et al. [88]. Briefly, ELECTRE II is based on two preference graphs representing the strong preferences (high  $p$  and low  $q$ ) and the weak preferences (lower  $p$  and higher  $q$ ). The weak preferences can be viewed as lower bounds on system performance that the decision maker is

willing to accept. Recently, versions III and IV of ELECTRE have been developed, but apparently the only references available on them are in French [89] [90], considerably limiting their audience.

ELECTRE has been applied to a substantial amount of practical problems with a predetermined finite set of alternatives evaluated in terms of ordinal (quantitative or qualitative) criteria, and could be most useful to solve multiobjective optimization problems that have those characteristics, since the technique is robust, simple, requires little input from the decision maker, and usually leads to plausible results.

More information on this method can be found in Roy [91] [20], Roy and Bertier [92] and Goicoechea et al. [51].

## 2.11 Multicriterion Polyhedral Dynamics or Q-Analysis

This method (also known as MCPD for its acronym) is similar to ELECTRE in the sense that it can also be applied only when there is available a discrete set of alternatives with respect to a set of criteria, and it can incorporate qualitative data into the analysis [93] [94]. The mathematical tool used in Q-analysis is algebraic topology, also called the polyhedral dynamics; hence, the name multicriterion polyhedral dynamics (MCPD) given to a new version of the technique [95]. In this technique the geometric structure at a multidimensional object called simplicial complex is considered. A relationship called the  $q$ -connection is defined on the elements (or simplices) of the complex, leading to an ordering imposed over that set of simplices [96].



The alternatives, which are found to have high  $q$ -levels, i.e., which satisfy a high proportion of criteria to an adequate level, are preferred to those having lower  $q$ -levels. In other words, systems are ranked according to the strength or ‘tightness’ of the multidimensional relationship between the set of criteria and the set of alternative systems. The MCPD technique is extremely efficient computationally speaking, and could certainly be substituted for ELECTRE provided an index which measures the discordance aspects of the problem is introduced. However, the technique is not always easy to visualize, because all non-trivial geometrical representations occur in  $n$ -dimensional Euclidean spaces with  $n > 3$ .

For more information on this method refer to Duckstein and Kempf [94] and Pfaff and Duckstein [95].

## 2.12 PROMETHEE

The PROMETHEE methods (Preference Ranking Organization METHod for Enrichment Evaluations) belong to the family of outranking methods (i.e., ELECTRE) introduced by B. Roy. These methods include two phases [97]:

- The construction of an outranking relation on the different criteria or objectives of the problem.
- The exploitation of this relation in order to give an answer to the multicriteria optimization problem.

In the first phase, a valued outranking relation based on a generalization of the notion of criterion is considered: a preference index is defined and a valued outranking graph, representing the preferences of the decision maker, is obtained.

The exploitation of the outranking relation is realized by considering for each action a leaving and an entering flow in the valued outranking graph: a

partial preorder (PROMETHEE I) or a complete preorder (PROMETHEE II) on the set of possible actions can be proposed to the decision maker in order to achieve the decision problem.

Brans criticizes the ELECTRE methods because they require too many parameters, the values of which are to be fixed by the decision maker and the analyst. They argue that even though some of these parameters have a real economic meaning and can, therefore, be fixed clearly, some others (such as concordance discrepancies and discrimination thresholds) playing an essential role in the procedures only have a technical character and their influence on the results is not always well understood. Moreover, in some of the ELECTRE methods, the notion of “degree of credibility” is rather difficult for practitioners [98].

In the PROMETHEE methods, Brans proposes an approach that is “very simple and easy to understand by the decision maker” according to him. These methods are based on extensions of the notion of criterion. These extended criteria can be easily built by the decision maker because they represent the natural notion of intensity of preference, and the parameters to be fixed (maximum 2) have a real economic meaning.

More information on this method may be found in Brans and Vincke [98], D’Avignon et al. [99], Dujardin [100], Mareschal and Brans and Brans [101] et al. [97].

More recently, Huylenbroeck [102] proposed the so-called **Conflict Analysis Model**, that combines the preference function approach of ELECTRE and PROMETHEE with the conflict analysis test of a method called ORESTE [103].

## 2.13 Dynamic Compromise Programming

The previous techniques are intended for static multiobjective problems in which the parameters do not change over time. However, in real-world applications is sometimes the case that variables such as cost, reliability, weight of a certain element, etc. change over time. Thus, the problem is said to be dynamic, and a new formulation of the multiobjective optimization problem has to be made, namely:

$$\min \sum_t \bar{z}(\bar{x}(t), \bar{s}(t)), \quad (2.49)$$

subject to

$$\begin{aligned} \bar{x}(t) &\in X, \quad \bar{s}(t) \in S \\ \bar{s}(t+1) &= \bar{H}(\bar{x}(t), \bar{s}(t)), \end{aligned} \quad (2.50)$$

where  $\bar{x}(t) \quad t = 1, 2, \dots, T$  is a time policy,  $T$  is the number of stages in which the system is divided,  $\bar{s}(t)$  is the state variable, and the function  $\bar{H}(\cdot)$  giving the state at time  $(t+1)$  as a function of the decision  $\bar{x}(t)$  or state  $\bar{s}(t)$  is called the **state transition function**.

It may be noted that  $t$  represents the construction stage rather than, strictly speaking, time.

There are several approaches to deal with the problem formulated in Equation (2.50): We can use classical dynamic programming [104] [105] and, by the use of the  $\varepsilon$ -constraint method, arrive at a set of efficient solutions [106] [107] [108] [109]. Other approaches are the consideration of utility functions over time [110] [111], preference order dynamic programming [112], and the nested Lagrangian multiplier method [113]. In the latter approach, elements of mathematical programming and decision analysis are integrated in a two-layer analysis. First, a

utility function is assessed subjectively at each stage of the process, and secondly, a subset of non-dominated solutions is found by mathematical programming.

An extension of compromise programming to dynamic problems seems to provide a promising dynamic multiobjective technique. Two algorithms which have been developed for this purpose [114] [115] lead to dynamic compromise solutions.

The idea of the method is to transform the original dynamic multiobjective problem into a classical dynamic programming problem with, however, a higher dimensional state vector. More generally, in Szidarovszky and Duckstein [116] a framework labelled RHOD has been developed to include generalized distance-based multiobjective approaches, compromise programming, goal programming, and cooperative game theory, as well as some non-distance based techniques, in particular the  $\varepsilon$ -constraint and the weighting methods. There is also a dynamic version of this framework, called DYRHOD. By transformation of variables, a dynamic program with  $k$  objectives is transformed into a single objective dynamic program with, however  $k - 1$  or at most  $k$  additional state variables. In the case when numerical difficulties occur because of the dimensionality problem, the use of differential dynamic programming [117] [118] turns out to be helpful [119].

## 2.14 PROTRADE

In this case, it is assumed that our multiobjective optimization problem has a probabilistic objective function and probabilistic constraints [120]. According to a 12-step algorithm, an initial solution is found using a surrogate objective function, then a multiattribute utility function is formed leading to a new surrogate objective function and a new solution. The solution is checked to see if it

is satisfactory to the decision maker. The process is repeated until a satisfactory solution is reached, as described in Goicoechea et al. [121] [51].

The results of the multiobjective optimization provide not only levels of attainment of the objective function elements, but also the probabilities of reaching those levels. The technique is interactive, which means that the decision maker formulates a preference function in a progressive manner, after a trial process [5].

Other stochastic methods are the **expected utility maximization** [122], and the so-called **multiobjective statistical method** [123] which is an extension of the surrogate trade-off method.

More information on this method may be found in Goicoechea et al. [124] [51].

## 2.15 STEP Method (STEM)

This is an iterative technique based on the progressive articulation of preferences. The basic idea is to converge toward the ‘best’ solution in the min-max sense, in no more than  $k$  steps, being  $k$  the number of objectives. This technique, which is mostly useful for linear problems, starts from an ideal point and proceeds in six steps, as summarized by Cohon [33]:

1. Construct a table of marginal solutions (strictly non-dominated if unique), by optimizing each objective function separately.
2. Compute, for each objective:

$$\alpha(i) = \frac{M(i) - m(i)}{M(i)} \left[ \sum_{j=1}^J c(i, j) \right]^2, \quad (2.51)$$

where

$M(i) = \max f_i(\bar{x})$ ,  $m(i) = \min f_i(\bar{x})$ , and  $c(i, j) = \text{cost coefficient of } i\text{th linear objective}$ .

Let the iteration index  $k = 0$

3. Compute  $\Pi(i) = \alpha(i) / \sum \alpha(i)$  and solve the min-max problem. Call the solution  $x(k)$ .
4. Show the solution to the decision maker:
  - (a) if satisfied, STOP;
  - (b) if not satisfied and  $k < p - 1$ , go to Step 5;
  - (c) if not satisfied and  $k > p - 1$ , STOP. A different procedure or at least a redefinition of the problem is required.
5. The decision maker selects an objective satisfied by the solution and determines the amount by which it can be decreased in order to improve the other objectives. If this cannot be done, some other approach is again required.
6. Define a new constraint relaxing the objective selected in Step 5. Set  $\alpha(i) = 0$  for that objective, increment  $k$  by one, and go to Step 3.

One criticism to this technique is the fact that it assumes that a best-compromise solution does not exist if it is not found after the  $k$  steps that the iterative process above described was executed. This does not give any clue to the decision maker of what to do [22]. Another problem is that it does not explicitly capture the trade-offs between the objectives. The weights in no way reflect a value judgment on the part of the decision maker. They are artificial quantities, generated by the analyst to reflect deviations from an ideal solution, which is itself

an artificial quantity. This definition of the weights serves to obscure rather than capture the normative nature of the multiobjective optimization problems [22].

More details of this technique may be found in Cohon and Marks [22], Szidarovszky and Duckstein [6], Ignizio [25] and Benayound et al. [125]. Other methods that also rely on the progressive articulation of preferences have been proposed by Klahr [126], Savir [127], Maier-Rothe and Stankard [128], Belenson and Kapur [129] and Monarchi et al. [130].

## 2.16 The Method of Zionts-Wallenius

Zionts and Wallenius [131] proposed a method which makes use of an implicit utility function on an interactive basis. Concavity of the objective functions and convexity of the constraint set are assumed. Nonlinear objective functions and constraints are first linearized. Thus, the method is dependent on the feasibility of using linear approximations to represent the constraint set and objective functions. The method assumes that the implicit utility function is a linear function of the objective functions.

Under this assumption, the best-compromise solution will be one of the extreme points solutions of the linearized constraint set. To start, a linear composite objective function is formed from a set of arbitrarily chosen positive weights. Optimization of this objective function identifies a nondominated extreme point solution. As this extreme point may not be the best compromise solution, adjacent nondominated points are presented to the decision maker for consideration. This is accomplished by identifying those nonbasic variables which would lead to nondominated solutions if brought into the basis. For each such variable, its introduction would simultaneously increase some objectives while reducing others. The tradeoffs for each nonbasic variable are presented to the decision maker.

The decision maker determines whether the tradeoffs are desirable, undesirable, or neither. Based on these responses, a new set of consistent weights is a nonzero amount and the current extreme solution would dominate the new one. Hence, nonbasic variables of this type can be discarded.

To decide whether the remaining nonbasic variables qualify for consideration, the following approach is used. First, solve the following sequence of linear programming problems for each remaining nonbasic variable:

$$\min f_r = \sum_{i=1}^p w_{ir} \lambda_i \quad (2.52)$$

subject to

$$\sum_{i=1}^p w_{ij} \lambda_i \geq 0, \quad j \in N, j \neq r \quad (2.53)$$

$$\sum_{i=1}^p \lambda_i = 1, \quad \lambda_i \geq 0, \quad i = 1, 2, \dots, p \quad (2.54)$$

where  $N$  = the set of nonbasic variables that have not been declared dominated.

Next, these tests are applied:

- **Test 1** : If  $f_r < 0$ , then nonbasic variable  $x_r$  leads to a nondominated solution.
- **Test 2** : If  $f_r \geq 0$ , then  $x_r$  does not lead to a nondominated solution.

The next step in the algorithm involves interaction with the decision maker. Let  $N_1$  represent the set of nonbasic variables satisfying Test 1. For each  $x_j \in N_1$  the decision maker is asked whether the tradeoff vector  $\bar{w}_j$  is desirable, undesirable, or neither. The decision maker responds, in effect, yes, no, or indifferent to the



trade. For positive, negative, and indifferent responses, we construct, respectively, constraints of the form

$$\sum_{i=1}^p w_{ij} \lambda_i \leq -\varepsilon \quad (2.55)$$

$$\sum_{i=1}^p w_{ij} \lambda_i \geq \varepsilon \quad (2.56)$$

$$\sum_{i=1}^p w_{ij} \lambda_i = 0 \quad (2.57)$$

where  $\varepsilon$  is a sufficiently small positive number.

The final step of the algorithm consists of finding a set of weights,  $\lambda^k$  that are consistent with the decision maker's tradeoff preferences. Then the process is repeated. The new set of weights is calculated by finding a feasible solution to constraints (2.54), (2.55), (2.56) and (2.57) by using linear programming. Each iteration uses the constraints (2.55) to (2.57) defined in the previous iteration plus the new ones added in the current iteration. The algorithm terminates when  $N_1$  is empty or when all  $x_j \in N_1$  represent an unattractive tradeoff by the decision maker. Since there is a finite number of extreme points and at least one extreme point is eliminated in each iteration, convergence occurs eventually.

More information on this method can be found in Goicoechea et al. [51].

## 2.17 Sequential Multiobjective Problem Solving Method (SEMOPS)

This method was proposed by Monarchi, Kisiel and Duckstein [130], and it basically involves the decision maker in an interactive fashion in the search for a satisfactory course of action.

A surrogate objective function is used based on the goal and aspiration levels of the decision maker. The goal levels are conditions imposed on the decision maker by external forces, and the aspiration levels are attainment levels of the objectives which the decision maker personally desires to achieve. One would say, then, that goals do not change once they are stated, but that the aspiration levels may change during the iteration process. The development of the algorithm will now be summarized [51]:

The decision problem consists of  $p$  goals,  $n$  decision variables, and a constraint set  $\bar{X}$ . Associated with each of the goals is an objective function which can be used to predict goal attainment or nonattainment. We write the set of all  $p$  objective functions as  $\bar{z} = (z_1, z_2, \dots, z_p)$ , and we use them to judge how well we have achieved our  $p$  goals. The range of the  $i$ th element of  $\bar{z}$  will be denoted by  $\Lambda z_i = [z_{iL}, z_{iu}]$ , which is not necessarily defined by the maximum and minimum values of the  $i$ th objective function. It is required that  $\bar{X}$  be continuous and that all objective and constraint functions be at least first-order differentiable. Thus the constraint or objective functions may be nonlinear. Nondimensionality is achieved by transforming  $z_i(\bar{x})$  into  $y_i(\bar{x})$  with a range of values in the interval  $[0, 1]$  such that

$$y_i(\bar{x}) = \frac{z_i(\bar{x}) - z_{iL}}{z_{iu} - z_{iL}} \quad (2.58)$$

Similarly, if we let  $AL = (AL_1, AL_2, \dots, AL_p)$  denote the vector of aspiration levels, the transformation

$$A_i = \frac{AL_i - z_{iL}}{z_{iu} - z_{iL}} \quad (2.59)$$

can be used to define  $A_i$  with values in the range  $[0, 1]$ .

Monarchi [132] suggests the use of the following transformations:

1. **At most**

$$z_i(\bar{x}) \leq AL_i; \quad d_i = \frac{z_i(\bar{x})}{AL_i} = \frac{y_i(\bar{x})}{A_i} \quad (2.60)$$

2. **At least**

$$z_i(\bar{x}) \geq AL_i; \quad d_i = \frac{AL_i}{z_i(\bar{x})} = \frac{A_i}{y_i(\bar{x})} \quad (2.61)$$

3. **Equals**

$$z_i(\bar{x}) = AL_i; \quad d_i = \frac{1}{2} \left[ \frac{AL_i}{z_i(\bar{x})} + \frac{z_i(\bar{x})}{AL_i} \right] = \frac{1}{2} \left[ \frac{A_i}{y_i(\bar{x})} + \frac{y_i(\bar{x})}{A_i} \right] \quad (2.62)$$

4. **Within an interval**

$$AL_{iL} \leq z_i(\bar{x}) \leq AL_{iU}; \quad d_i = \left[ \frac{AL_{iU}}{AL_{iL} + AL_{iU}} \right] \left[ \frac{AL_{iL}}{z_i(\bar{x})} + \frac{z_i(\bar{x})}{AL_{iU}} \right] \quad (2.63)$$

In each instance, values of  $d_i \leq 1$  imply that the  $i$ th objective is satisfied. We note also that, except for the first type, the  $d_i$  are nonlinear functions of the  $i$ th objective.

Operationally, SEMOPS is a three-step procedure involving setup, iteration, and termination. Setup involves structuring a principal problem and a set of

auxiliary problems with surrogate objective function. The iteration step involves cycling between an optimization phase (by the analyst), and an evaluation phase (by the decision maker) until a satisfactory solution is reached, if it exists. The procedure terminates when either a satisfactory solution is found, or the decision maker concludes that none of the nondominated solutions obtained are satisfactory and gives up in the search.

For the first iteration, then, the principal problem and set of  $P$  auxiliary problems shown below are solved:

Principal problem

$$\min s_1 = \sum_{i=1}^P d_i \quad (2.64)$$

subject to

$$\bar{x} \in \bar{X} \quad (2.65)$$

and the set of auxiliary problems,  $l = 1, 2, \dots, P$ .

$$\min s_{1l} = \sum_{i=1, i \neq l}^P d_i \quad (2.66)$$

subject to

$$\bar{x} \in \bar{X} \quad (2.67)$$

$$z_l(\bar{x}) \geq AL_l \quad (2.68)$$

The resulting solutions are used in the evaluation process to assist the decision maker to determine the “direction of change” for the next iteration.

More information on this method can be found in Goicoechea et al. [51].

## 2.18 Local Multiattribute Utility Functions

When I talked before about multiattribute utility theory, I mentioned that we can distinguish two different approaches for utility maximization: the global and the local. The first approach uses assumptions that limit the form of the utility function to specific families of curves. Once the family is selected, a few assessments determine the free parameters. The resulting utility function applies over the entire decision region. The main drawback of this approach is that the additional assumptions that are required to derive these functions are reasonable in local regions, in the small, but when assumed globally, in the large, they are very restrictive. This implies that the decision maker will be forced sometimes to fit a function not truly representing his/her preferences.

On the other hand, local utility maximization provides an alternative approach that avoids restrictive assumptions. Instead of specifying the utility function in the large, a sequence of local linear approximations of the utility function are built in the small. Each linear approximation yields a trial solution. Under appropriate conditions each trial solution is preferred to its predecessor, so the trial sequence eventually reaches the optimum. This iterative technique avoids the strong restrictions, but usually requires a large number of assessments. Since each assessment requires a time-consuming interaction with the decision maker, the local procedures are normally not used in practice.

Oppenheimer [76] introduced a method which merges the global and local procedures into an algorithm that tries to incorporate the best of both approaches. He calls his/her technique the *proxy approach*, considering that a *proxy attribute* is one that reflects the degree to which an associated objective is met but does

not directly measure the objective. In other words, a proxy attribute indirectly measures the achievement of a certain objective.

The basic idea of Oppenheimer is to use the sum-of-exponentials and the sum-of-powers utility functions as local proxies in an interactive feasible directions algorithms. As a first step, he uses the global assessment procedure to encode the utility function; but instead of using this function as a global model, it is used only as a local proxy. At each iteration, a new tradeoff vector is assessed to update the proxy. Since the utility function is a very good model in the small, the new proxy algorithm converges at a much higher rate. Oppenheimer never assumes that the proxy is the true utility function, even in the small, but he only uses it as a mechanism to guide the search for the optimal decision. To ensure convergence of this algorithm the utility function must be concave, the feasible region must be convex, and a feasible directions methods must be used.

At first sight, this method seems to be much more expensive than normal local procedures, since the information requirements of the proxy seems to add a considerable assessment burden. However, the extra information required is already available, since tradeoffs are assessed at each iteration, and the past information generated by the global method is not discarded, but instead it is used. Therefore, there is no additional information cost involved.

Also, since the conventional linear approximations used in the local approach are replaced by a nonlinear proxy, one could expect each maximization to be more complicated, since each iteration would require a nonlinear rather than a linear program. However, the sum-of-exponentials and sum-of-powers utility functions developed by Oppenheimer have a special mathematical structure that simplifies the task. Both are concave and separable, so the maximization with linear constraints is a concave programming problem. Concave separable techniques, using

a series of linear programs, make this problem relatively easy to solve. This makes this technique also very efficient, computationally speaking.

In summary, we can say that Oppenheimer's algorithm uses the advantages of one technique to overcome the disadvantages of the other, and the result is a powerful combined technique that yields rapid convergence without restrictive assumptions. The technique seems very suitable for decision making under certainty, but there seem to be major obstacles to handle decision problems under uncertainty [133].

For more information on this method, refer to Oppenheimer [133] [76].

## 2.19 The Method of Nijkamp and Vos

This method is a variant of the concordance analysis, or ELECTRE method that I mentioned before, which is based on a pairwise comparison of (weighted) project outcomes, and it can be used both as an elimination method for less desirable projects and as a selection method of good projects. This new variant, developed by Nijkamp and Vos [134] is based on the idea of satisficing (or norm) project outcomes, which may serve as a frame of reference for the evaluation techniques. The use of norm outcomes allows the evaluation analysis to be carried out in terms of relative deviations between the actual and the norm outcomes, so that the problems caused by different scales (dimensions) can be attacked simultaneously.

The necessary and sufficient condition for an optimal plan to exist in this method, is that this plan is not dominated by any other plan. If there is not a single non-dominated alternative, then two possibilities with which to arrive at a more definite conclusion can be distinguished:

- Change the threshold values of the concordance and discordance index. These indexes reflex the relative importance of each plan with respect to each other, and in this technique, they have to be adapted in order to be able to use the norm outcomes.
- Add a new selection criterion (for example, a maximal average concordance index and a minimal discordance index for a certain plan).

In this way a unique consistent solution can always be found, although the robustness of this solution is lower as more relaxations of threshold values are carried out.

The main advantage of concordance analysis is that decisions can be taken on the basis of multidimensional criteria. A drawback is the determination of the weights, although the weighting schemes offer more opportunities for interactive communication with decision makers.

The stability of an optimal plan can be analyzed by calculating critical values for the weights or project outcomes. Beyond these critical values the original optimal plan will be replaced by an alternative plan. Another possibility for checking the stability of an optimal plan is stochastic analysis, in which a probability distribution function for the weights and (some uncertain) project outcomes is specified. This stochastic analysis is particularly useful if decision makers are not able to specify the weights accurately.

For more information on this method, refer to Nijkamp and Vos [134].



## 2.20 The Nested Lagrangian Multiplier Method (NLM)

The main idea in this method is to be able to solve multidimensional optimization problems without comprehensively deriving the Pareto frontier. The core of this method, developed by Seo and Sakawa [113] is to form a hierarchical modeling of the multilevel systems, and use Lagrangian multipliers to derive utility functions.

The original multicriteria optimization problem is divided into two layers. In the first one, we have a set of scalar optimization problems, and we apply mathematical programming to each one. Therefore, we will be optimizing single-objective functions. At the second layer, these scalar optimization problems are coordinated into an overall system: optimal solutions obtained from the first layer optimization processes are combined with a weighting method. The main problem then, becomes to find a more sophisticated weighting method. With that in mind, Seo and Sakawa proposed the use of a nesting algorithm that uses Lagrangian multipliers, or a dual optimal solution obtained from mathematical programming as a medium to derive component utility functions, so that multiattribute utility theory may be used.

Mathematical programming is also considered in a hierarchical structure for each subsystem or scalar optimization problem. Namely, an objective function of mathematical programming represents a “lower level” objective peculiar to each subsystem. Constraint constants are regarded as “upper level” objectives which are sent from the “upper-level” decision maker. Decision variables are a normative instrument for achieving these objectives and regarded as the lowest level objectives. Thus, formulations of mathematical programming are considered

in the framework of a hierarchical systems structure. It is precisely this the main distinction between NLM and the surrogate worth trade-off method, since whereas in the first the worth assessment is directly given to the Lagrangian multipliers in optimal, the second the Lagrangian multipliers are interpreted as the trade-off rate functions between the objectives, and the worth assessment is indirectly given to the trade-off rate functions instead of to the objective functions.

The effectiveness of this method depends on computational efficiency in solving problems of mathematical programming. In the case of large-scale problems, this could be an issue, mainly when convexity cannot be insured, since convergence of nonlinear programming will be difficult in such cases. Nevertheless, it is an interesting proposal which combines decision making with mathematical programming in a very rational way.

For more information on this method, refer to Seo and Sakawa [113].

## 2.21 Rao's Method for Fuzzy Systems

Rao [135] considers the case in which we have a multi-objective optimization system in which there is a vast amount of fuzzy information in both the objective and the constraint functions. He proposes a methodology with which a multi-objective fuzzy structural optimization problem is transformed into an ordinary single-objective optimization problem.

In the multi-objective optimization problem, we need the satisfaction of the objective function 'and' the constraints; therefore a decision or selection of a set of design variables in a fuzzy environment can be made by assuming that the constraints are 'non-interactive' and that the logical 'and' corresponds to the intersection. The decision in a fuzzy environment can therefore be viewed as the intersection of the fuzzy constraints and the fuzzy objective function. An

important feature of fuzzy set theory is the symmetry between objective functions and constraints [135].

The fuzzy optimization problem is stated by Rao [135] as:

$$\left. \begin{array}{l} \text{find } \bar{X} \text{ which minimizes } f(\bar{X}) \\ \text{subject to } g_i(\bar{X}) \in \tilde{G}_i, \quad i = 1, 2, \dots, m \end{array} \right\} \quad (2.69)$$

where the wave symbols denote that the variables or operations contain fuzzy information, and  $\tilde{G}_i$  indicates the allowable interval for the constraint function  $g_i$ . If  $d_i$  denotes the permissible variation of  $g_i$ , then  $\tilde{G}_i = [-\infty, b_i + d_i]$ . The constraint  $g_i \in \tilde{G}_i$  means that  $g_i$  is a member of the fuzzy set  $\tilde{G}_i$  in the sense of  $\mu_{g_i} > 0$ , where  $\mu_{g_i}$  is the membership function (or degree of satisfaction), and it is defined according to the constraints of the problem such that it has a value between 0 and 1. For example, when applied to structural optimization,  $\mu_{g_i}$  could be defined as:

$$\mu_{g_i}(\bar{X}) = \begin{cases} 1, & \text{if } \sigma - \sigma^{(u)} \leq 0 \\ 0, & \text{if } \sigma - \sigma^{(u)} > d_i \end{cases} \quad (2.70)$$

being  $\sigma$  the stress of the structure, and  $\sigma^{(u)}$  its upper bound.

If  $\sigma - \sigma^{(u)}$  lies between 0 and  $d_i$ ,  $\mu_{g_i}(\bar{X})$  can be defined to have a value varying between 1 and 0.

The fuzzy feasible region is defined by considering all the constraints as

$$\tilde{S} \cap_{i=1}^m \tilde{G}_i \quad (2.71)$$

and the membership degree of any design vector  $\bar{X}$  to fuzzy feasible region  $\tilde{S}$  is given by

$$\mu_{\tilde{S}}(\bar{X}) = \min_i \{\mu_{g_i}(\bar{X})\} \quad (2.72)$$

that is the minimum degree of satisfaction of the design vector  $\bar{X}$  to all of the constraints. A design vector  $\bar{X}$  can be considered feasible if  $\mu_{\tilde{S}}(\bar{X}) > 0$  and the objective function defines a fuzzy domain  $D$  in  $\tilde{S}$  as

$$D = \{\mu_f(\bar{X})\} \cap \left\{ \bigcap_{i=1}^m \mu_{g_i}(\bar{X}) \right\} \quad (2.73)$$

From the fuzzy domain  $D$ , and optimum solution  $\bar{X}^*$  can be selected as the design for which the membership function is maximum:

$$\mu_D(\bar{X}^*) = \max \mu_D(\bar{X}) \quad (2.74)$$

where

$$\mu_D = \min \{\mu_f(\bar{X}), \mu_{g_1}(\bar{X}), \dots, \mu_{g_m}(\bar{X})\} \quad (2.75)$$

The fuzzy single-objective optimization approach can be generalized to fuzzy multi-objective optimization problems by defining the fuzzy domain corresponding to the objective functions and the constraints as

$$D = \left\{ \bigcap_{j=1}^k \mu_{f_j}(\bar{X}) \right\} \cap \left\{ \bigcap_{i=1}^m \mu_{g_i}(\bar{X}) \right\} \quad (2.76)$$

with

$$\mu_D = \min_{j,i} \{\mu_{f_j}(\bar{X}), \mu_{g_i}(\bar{X})\} \quad (2.77)$$

where  $\mu_{f_j}(\bar{X})$  and  $\mu_{g_i}(\bar{X})$  denote the membership functions of the  $j$ th objective and  $i$ th constraint functions, respectively. The optimum solution  $\bar{X}^*$  is selected such that

$$\mu_D(\bar{X}^*) = \max \mu_D(\bar{X}) \quad (2.78)$$

Computationally, the solution of the multi-objective fuzzy optimization problem indicated in equation(2.78) can be found by (i) finding the solutions of the individual single-objective optimization problems, (ii) determining the best and worst solutions possible for each of the objective functions, (iii) using these solutions as boundaries of the fuzzy ranges in the corresponding fuzzy optimization problem and (iv) solving the resulting fuzzy optimization problem. The details are indicated in the following step-by-step procedure [135]:

1. Starting from any trial design vector  $\bar{X}_s$ , minimize the individual objective function  $f_j(\bar{X})$  subject to the constraints  $g_i(\bar{X}) \leq b_i$ ,  $i = 1, 2, \dots, m$  using ordinary (crisp) optimization procedures. Let the solution be  $\bar{X}_j^*$ ,  $i = 1, 2, \dots, k$ .

2. Construct a matrix  $[\bar{P}]$  as

$$[\bar{P}] = \begin{bmatrix} f_1(\bar{X}_1^*) & f_2(\bar{X}_2^*) & \dots & f_k(\bar{X}_1^*) \\ f_1(\bar{X}_2^*) & f_2(\bar{X}_2^*) & \dots & f_k(\bar{X}_2^*) \\ \vdots & & & \\ f_1(\bar{X}_k^*) & f_2(\bar{X}_k^*) & \dots & f_k(\bar{X}_k^*) \end{bmatrix} \quad (2.79)$$

It can be seen that the diagonal elements in the matrix  $[\bar{P}]$  are the minima in their respective columns.

3. The minimum and maximum possible values of the objective functions are identified as

$$\left. \begin{aligned} f_j^{min} &= \min_i f_j(\bar{X}_i^*) = f_j(\bar{X}_j^*) \\ f_j^{max} &= \max_i f_j(\bar{X}_i^*) \end{aligned} \right\}, \quad j = 1, 2, \dots, k \quad (2.80)$$

4. From the extreme values of  $f_j$  determined in equation (2.80), the membership functions of the fuzzy objective functions are constructed as

$$\mu_{f_j}(\bar{X}) = \begin{cases} 0, & \text{if } f_j(\bar{X}) > f_j^{max} \\ \left( \frac{-f_j(\bar{X}) + f_j^{max}}{f_j^{max} - f_j^{min}} \right), & \text{if } f_j^{min} < f_j(\bar{X}) \leq f_j^{max}, \quad j = 1, 2, \dots, k \\ 1, & \text{if } f_j(\bar{X}) \leq f_j^{min} \end{cases} \quad (2.81)$$

5. The fuzzy constraints can be stated as

$$g_i(\bar{X}) \leq b_i + d_i, \quad i = 1, 2, \dots, m \quad (2.82)$$

where  $d_i$  denotes the distance by which the boundary of the  $i$ th constraint is moved. The membership function of the  $i$ th constraint can be defined as

$$\mu_{g_i}(\bar{X}) = \begin{cases} 0, & \text{if } g_i(\bar{X}) > b_i + d_i \\ 1 - \left\{ \frac{g_i(\bar{X}) - b_i}{d_i} \right\}, & \text{if } b_i \leq g_i(\bar{X}) \leq b_i + d_i, \quad i = 1, 2, \dots, m \\ 1, & \text{if } g_i(\bar{X}) < b_i \end{cases} \quad (2.83)$$

6. By considering the optimum solution as the intersection of the membership functions of the objective functions and constraints, the solution of the

fuzzy multi-objective optimization problem can be found by determining  $\bar{X}$  and  $\lambda$  which maximize  $\lambda$  subject to

$$\left. \begin{aligned} \lambda &\leq \mu_{f_j}(\bar{X}), \quad j = 1, 2, \dots, k \\ \lambda &\leq \mu_{g_i}(\bar{X}), \quad i = 1, 2, \dots, m \end{aligned} \right\} \quad (2.84)$$

This problem can be solved using ordinary single-objective non-linear programming techniques.

For more information on this topic, refer to Rao [135], Dhingra et al. [136] and Blin [137].

## 2.22 Displaced Ideal

Zeleny [41] tried to develop a methodology that emulates the criteria that drives a human being to adopt a certain solution, which he calls the **ideal point**. Coombs [138] introduced the idea of ideal point, and he showed that the probabilities of choice depend on whether alternatives being compared lie on the same side of the ideal point, or whether some lie on one side of the ideal and some on the other.

When there are constraints, it becomes an unrealistic goal to reach the ideal point, and there is a conflict between what is preferable and what is possible. To solve it, a decision must be taken by exploring the limits achievable along each particular attribute of importance, so that the **ideal alternative** may be defined.

General infeasibility or nonavailability of this ideal alternative creates a pre-decision conflict and generates the impulse to move “as close as possible” towards it. Because of the experienced conflict, the decision maker starts searching for new alternatives, preferably those which are the closest to the ideal one.

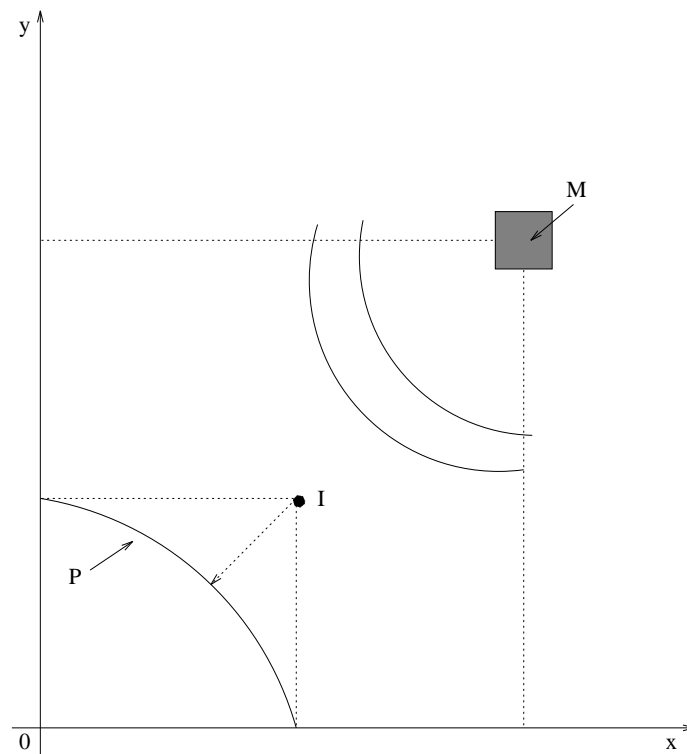


Figure 2.9: An example of a case where reaching the ideal point ( $M$ ) is an unrealistic goal, and we search, instead, an alternative point ( $I$ ).



If this ideal alternative is found, then there is no need for any further decision making process, because the conflict has been removed and the ideal has become feasible. It should be observed that, in contrast with the relative stability of the ideal point, the ideal alternative can be displaced quite frequently in dependency on changes in the available set, objectives, evaluations, measurements, etc. It becomes a moving target, a point of reference which provides a model for human adaptivity, intransitivity and dynamic adjustment of preferences. Then, decision making becomes the dynamic process of searching the ideal point via the ideal alternative. Figure 2.9 (taken from Zeleny [41]) shows an example of a case where reaching the ideal point ( $M$  in the graph) becomes an unrealistic goal, and instead an ideal alternative is searched ( $I$  in the graph).

Partial decisions taking may consist of discarding some “obviously” inferior alternatives, re-considering previously rejected alternatives, adding or deleting criteria, etc.

As all alternatives are compared with the ideal, those which are the farthest away are removed from further consideration. There are many important impacts of such partial decisions. First, whenever an alternative is discarded there could be a shift in a maximum available score to the next lower feasible level. Thus, the ideal alternative is being displaced **closer** to the feasible set. Similarly, addition of a new alternative may displace the ideal farther away. Such displacements induce changes in evaluations, attribute importance and ultimately in the preference ordering of remaining alternatives. All alternatives are then compared with respect to the new, displaced ideal, to take a final decision.

More information on this technique may be found in Zeleny [41] [139].

## 2.23 Lexicographic Method

In this method, the objectives are ranked in order of importance by the designer. The optimum solution  $\bar{X}^*$  is then obtained by minimizing the objective functions, starting with the most important one and proceeding according to the order of importance of the objectives.

Let the subscripts of the objectives indicate not only the objective function number, but also the priority of the objective. Thus,  $F_1(\bar{X})$  and  $F_k(\bar{X})$  denote the most and least important objective functions, respectively. Then the first problem is formulated as

$$\text{Minimize } F_1(\bar{X}) \quad (2.85)$$

subject to

$$g_j(\bar{X}) \leq 0; \quad j = 1, 2, \dots, m \quad (2.86)$$

and its solution  $\bar{X}_1^*$  and  $F_1^* = (F_1^*)$  is obtained. Then the second problem is formulated as

$$\text{Minimize } F_2(\bar{X}) \quad (2.87)$$

subject to

$$g_j(\bar{X}) \leq 0; \quad j = 1, 2, \dots, m \quad (2.88)$$

$$F_1(\bar{X}) = F_1^* \quad (2.89)$$

and the solution of this problem is obtained as  $X_2^*$  and  $F_2^* = F_2(X_2^*)$ . This procedure is repeated until all  $k$  objectives have been considered. The  $i$ th problem is given by

$$\text{Minimize } F_i(\bar{X}) \quad (2.90)$$

subject to

$$g_j(\bar{X}) \leq 0; \quad j = 1, 2, \dots, m \quad (2.91)$$

$$F_l(\bar{X}) = F_l^*, \quad l = 1, 2, \dots, i-1 \quad (2.92)$$

The solution obtained at the end, i.e.,  $X_k^*$  is taken as the desired solution  $X^*$  of the problem.

More information on this method may be found in Rao [52] and Sarma et al. [140].

## 2.24 Goal-Attainment Method

This method is not subject to convexity limitations of any kind. In this approach, a vector of weights  $w_1, w_2, \dots, w_k$  relating the relative under- or over-attainment of the desired goals must be elicited from the decision maker in addition to the goal vector  $b_1, b_2, \dots, b_k$  for the objective functions  $f_1, f_2, \dots, f_k$ . To find the best-compromise solution  $X^*$ , we solve the following problem:

$$\text{Minimize } \alpha \quad (2.93)$$

subject to:

$$\begin{aligned} g_j(\bar{X}) &\leq 0; \quad j = 1, 2, \dots, m \\ b_i + \alpha \cdot w_i &\geq f_i(\bar{X}); \quad i = 1, 2, \dots, k \end{aligned} \quad (2.94)$$

where  $\alpha$  is a scalar variable unrestricted in sign and the weights  $w_1, w_2, \dots, w_k$  are normalized so that

$$\sum_{i=1}^k |w_i| = 1 \quad (2.95)$$

If some  $w_i = 0$  ( $i = 1, 2, \dots, k$ ), it means that the maximum limit of objectives  $f_i(\bar{X})$  is  $b_i$ .

It can be easily shown [141] (taken from Chen and Liu [141]) that the set of non-inferior solutions can be generated by varying the weights, with  $w_i \geq 0$  ( $i = 1, 2, \dots, k$ ) even for nonconvex problems. The mechanism by which this method operates is illustrated in Figure 2.10. The vector  $\bar{b}$  is represented by the decision goal of the decision maker, who also decides the direction of  $\bar{w}$ . Given vector  $\bar{w}$  and  $\bar{b}$ , the direction of the vector  $\bar{b} + \alpha \cdot \bar{w}$  can be determined, and the problem stated by equation (2.93) is equivalent to finding a feasible point on this vector in objective space which is closest to the origin. It is obvious that the optimal solution of equation (2.93) will be the first point at which  $\bar{b} + \alpha \cdot \bar{w}$  intersects the feasible region  $F$  in the objective space. Should this point of intersection exist, it would clearly be a noninferior solution.

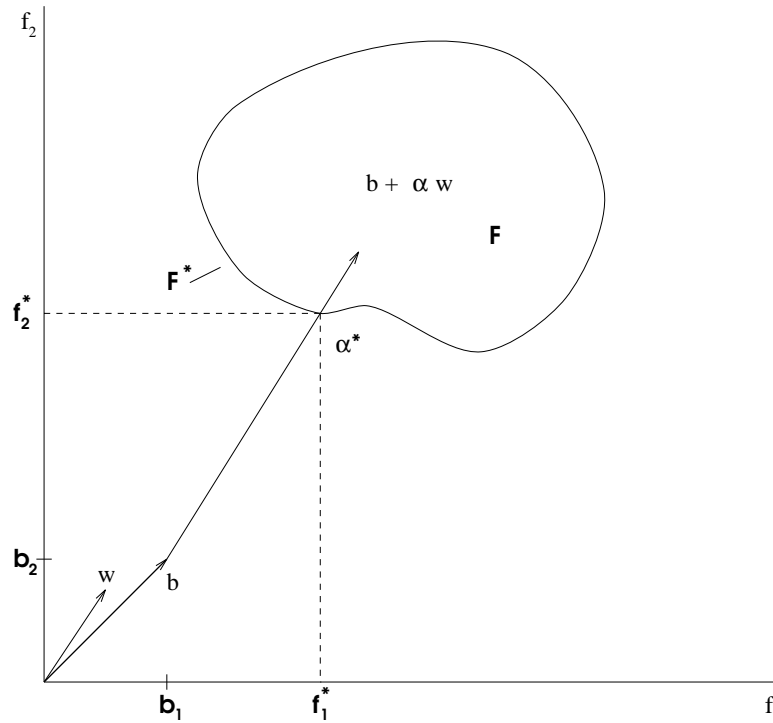


Figure 2.10: Goal-attainment method with two objective functions.

It should be pointed out that the optimum value of  $\alpha$  will inform the decision maker of whether the goals are attainable or not. A negative value of  $\alpha$  implies that the goal of the decision maker is attainable and an improved solution will be obtained. Otherwise, if  $\alpha > 0$ , then the decision maker goal is unattainable.’

For more information on this method, refer to Chen and Liu [141] and Rao [52].

There is an impressive amount of information on multiobjective optimization. In 1992, John Wiley and Sons started publishing a new journal specialized in this subject, with 3 issues per year, called **Journal of Multi-criteria Decision Analysis**. Jaap Hartog (mcrit@sjaan.fbk.eur.nl) initiated an electronic multi-criteria discussion list, called MCRIT-L, to serve as a discussion forum for sharing information about multi-criteria decision making in all its various aspects.

This should give an idea of the huge volume of information available on this subject. Therefore, this chapter is just a highly simplified general view of the field, that does not attempt, under any means, to cover all the existing multiobjective optimization methods.

## Chapter 3

# Multiobjective Optimization using Genetic Algorithms

### 3.1 A Gentle Introduction to Genetic Algorithms

The famous naturalist Charles Darwin defined *Natural Selection* or *Survival of the Fittest* in his book [142] as the *preservation of favorable individual differences and variations, and the destruction of those that are injurious*. In nature, individuals have to adapt to their environment in order to survive in a process called *evolution*, in which those features that make an individual more suited to compete are preserved when it reproduces, and those features that make it weaker are eliminated. Such features are controlled by units called **genes** which form sets called **chromosomes**. Over subsequent generations not only the fittest individuals survive, but also their fittest genes which are transmitted to their descendants during the sexual recombination process which is called **crossover**.

John H. Holland became interested in the application of natural selection to machine learning, and in the late 60s, while working at the University of Michigan, he developed a technique that allowed computer programs to mimic the process of evolution. Originally, this technique was called **reproductive plans**, but the

term **genetic algorithm** became popular after the publication of his book [143] [144].

In 1989, Goldberg published a book [145] that provided a solid scientific basis for this area, and cited no less than 73 successful applications of the genetic algorithm. In the last few years the growing interest on this technique is reflected in a larger number of conferences, a new international journal, and an increasing amount of software and literature devoted to this subject.

Koza [146] provides a good definition of a GA:

The **genetic algorithm** is a highly parallel mathematical algorithm that transforms a set (**population** of individual mathematical objects (typically fixed-length character strings patterned after chromosome strings), each with an associated **fitness** value, into a new population (i.e., the next **generation**) using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations (notably sexual recombination).

Actually, the genetic algorithm derives its behavior from a metaphor of one of the mechanisms of evolution in nature which is called **hard selection** [147]. Under this scheme, only the best available individuals are retained for generating descendants. This contrasts with **soft selection**, which offers a probabilistic mechanism for maintaining individuals to be parents of future progeny despite possessing relatively poorer objective values.

It has been argued [147] that the term **genetic algorithm** (GA) is misleading, since natural selection is only one of the mechanisms of evolution, and it would be more appropriate to call them **hard selection** (HS) algorithms to



reflect the fact that they deal with only that particular selection scheme. However, the term is so common today, that a change does not seem feasible, at least in the near future.

A genetic algorithm for a particular problem must have the following five components [148]:

1. A representation for potential solutions to the problem.
2. A way to create an initial population of potential solutions.
3. An evaluation function that plays the role of the environment, rating solutions in terms of their “fitness”.
4. Genetic operators that alter the composition of children.
5. Values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

Some of the basic terminology used by the genetic algorithms (GAs) community is the following [147]:

- A **chromosome** is a data structure that holds a “string” of task parameters, or genes. This string may be stored, for example, as a binary bit-string (binary representation) or as an array of integers (floating point or real-coded representation) that represent a floating point number. This chromosome is analogous to the base-4 chromosomes present in our own DNA. Normally, in the GA community, the haploid model of a cell is assumed (one-chromosome individuals). However, diploids have also been used in the past [145].
- A **gene** is a subsection of a chromosome that usually encodes the value of a single parameter.

- An **allele** is the value of a gene. For example, for a binary representation each gene may have an allele of 0 or 1, and for a floating point representation, each gene may have an allele from 0 to 9.
- A **schema** (plural **schemata**) is a pattern of gene values in a chromosome, which may include “do not care” states (represented by a # symbol). Thus in a binary chromosome, each schema can be specified by a string of the same length as the chromosome, with each character being one of { 0, 1, # }. A particular chromosome is said to “contain” a particular schema if it matches the scheme (e.g. chromosome 01101 matches schema #1#0#).
- If the solution of a problem can be represented by a set of  $N$  real-valued parameters, then the job of finding this solution can be thought of as a search in an  $N$ -dimensional space. This region is simply referred as the **search space** of the problem.
- The **fitness** of an individual is a value that reflects its performance (i.e., how well solves a certain task). A **fitness function** is a mapping of the chromosomes in a population to their corresponding fitness values. A **fitness landscape** is the hypersurface obtained by applying the fitness function to every point in the search space.
- A **building block** is a small, tightly clustered group of genes which have co-evolved in such a way that their introduction into any chromosome will be likely to give increased fitness to that chromosome. The **building block hypothesis** [145] states that GAs generate their solutions by first finding as many building blocks as possible, and then combining them together to give the highest fitness.

- **Deception** is a condition under which the combination of good building blocks leads to reduced fitness, rather than increased fitness. This condition was proposed by Goldberg [145] as a reason for the failure of GAs on certain tasks.
- **Elitism** (or an elitist strategy) is a mechanism which ensures that the chromosomes of the highly fit member(s) of the population are passed on to the next generation without being altered by any genetic operator. The use of elitism guarantees that the maximum fitness of the population never decreases from one generation to the next, and it normally produces a faster convergence of the population.
- **Epistasis** is the interaction between different genes in a chromosome. It is the extent to which the contribution to fitness of one gene depends on the values of other genes. Geneticists use this term to refer to a “masking” or “switching” effect among genes, and a gene is considered to be “epistatic” if its presence suppresses the effect of a gene at another locus. This concept is closely related to deception, since a problem with high degree of epistasis is deceptive, since building blocks can not be formed. On the other hand, problems with little or no epistasis are trivial to solve (hill climbing is sufficient).
- **Exploitation** is the process of using information gathered from previously visited points in the search space to determine which places might be profitable to visit next. Hill climbing is an example of exploitation, because it investigates adjacent points in the search space, and moves in the direction giving the greatest increase in fitness. Exploitation techniques are good at

finding local minima (or maxima). The GA uses crossover as an exploitation mechanism.

- **Exploration** is the process of visiting entirely new regions of a search space, to see if anything promising may be found there. Unlike exploitation, exploration involves leaps into unknown regions. Random search is an example of exploration. Problems which have many local minima (or maxima) can sometimes only be solved using exploration techniques such as random search. The GA uses mutation as an exploration mechanism.
- A **genotype** represents a potential solution to a problem, and is basically the string of values chosen by the user, also called chromosome.
- A **phenotype** is the meaning of a particular chromosome, defined externally by the user.
- Genetic drift is the name given to the changes in gene/allele frequencies in a population over many generations, resulting from chance rather than from selection. It occurs most rapidly in small populations and can lead to some alleles to become extinct, thus reducing the genetic variability in the population.
- A **niche** is a group of individuals which have similar fitness. Normally in multiobjective and multimodal optimization, a technique called **sharing** is used to reduce the fitness of those individuals who are in the same niche, in order to prevent the population to converge to a single solution, so that stable sub-populations can be formed, each one corresponding to a different objective or peak (in a multimodal optimization problem) of the function.

The basic operation of a Genetic Algorithm is illustrated in the following segment of pseudo-code [149]:

```

generate initial population, G(0);
evaluate G(0);
t:=0;
repeat
    t:=t+1;
    generate G(t) using G(t-1);
    evaluate G(t);
until a solution is found

```

First, an initial population is randomly generated. The individuals of this population will be a set of chromosomes or strings of characters (letters and/or numbers) that represent all the possible solutions to the problem. We apply a **fitness function** to each one of these chromosomes in order to measure the quality of the solution encoded by the chromosome. Knowing each chromosome's fitness, a **selection** process takes place to choose the individuals (presumably, the fittest) that will be the parents of the following generation. The most commonly used selection schemes are the following [150]:

- **Proportionate Reproduction:** This term is used generically to describe several selection schemes that choose individuals for birth according to their objective function values  $f$ . In these schemes, the probability of selection  $p$  of an individual from the  $i$ th class in the  $t$ th generation is calculated as

$$p_{i,t} = \frac{f_i}{\sum_{j=1}^k m_{j,t} f_j} \quad (3.1)$$

where  $k$  classes exist and the total number of individuals sums to  $n$ . Several methods have been suggested for sampling this probability distribution, including Monte Carlo or **roulette wheel** selection [151], **stochastic**

**remainder** selection [152] [153], and **stochastic universal** selection [154] [155].

- **Ranking Selection:** In this scheme, proposed by Baker [156] the population is sorted from best to worst, and each individual is copied as many times as it can, according to a non-increasing assignment function, and then proportionate selection is performed according to that assignment.
- **Tournament Selection:** The population is shuffled and then is divided into groups of  $k$  elements from which the best individual (i.e., the fittest) will be chosen. This process has to be repeated  $k$  times because on each iteration only  $m$  parents are selected, where

$$m = \frac{\text{population size}}{k}$$

For example, if we use binary tournament selection ( $k = 2$ ), then we have to shuffle the population twice, since in each stage half of the parents required will be selected. The interesting property of this selection scheme is that we can guarantee multiple copies of the fittest individual among the parents of the next generation.

After being selected, **crossover** takes place. During this stage, the genetic material of a pair of individuals is exchanged in order to create the population of the next generation. The two main ways of performing crossover are called single-point and two-point crossover. When a **single-point** crossover scheme is used, a position of the chromosome is randomly selected as the crossover point as indicated in Figure 3.1. When a **two-point** crossover scheme is used, two positions of the chromosome are randomly selected as indicated in Figure 3.2.

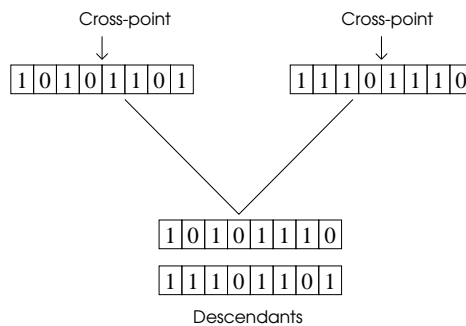


Figure 3.1: Use of a single-point crossover between two chromosomes. Notice that each pair of chromosomes produces two descendants for the next generation. The cross-point may be located at the string boundaries, in which case the crossover has no effect and the parents remain intact for the next generation.

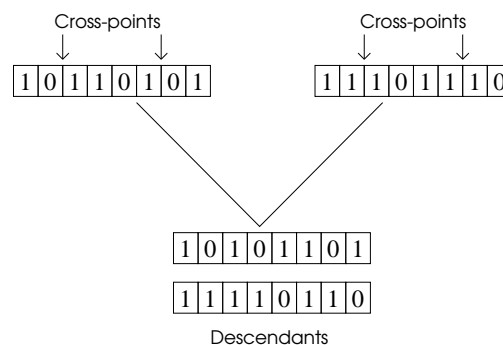


Figure 3.2: Use of a two-point crossover between two chromosomes. In this case the genes at the extremes are kept, and those in the middle part are exchanged. If one of the two cross-points happens to be at the string boundaries, a single-point crossover will be performed, and if both are at the string boundaries, the parents remain intact for the next generation.

**Mutation** is another important genetic operator that randomly changes a gene of a chromosome. If we use a binary representation, a mutation changes a 0 to 1 and viceversa. This operator allows the introduction of new chromosomal material to the population and, from the theoretical perspective, it assures that—given any population—the entire search space is connected [149].

If we knew in advance the final solution, it would be trivial to determine how to stop a genetic algorithm. However, as this is not normally the case, we have to use one of the two following criteria to stop the GA: either give a fixed number of generations in advance, or verify when the population has stabilized (i.e., all or most of the individuals have the same fitness).

GAs differ from traditional search techniques in several ways [149]:

- GAs do not require problem specific knowledge to carry out a search.
- GAs use stochastic instead of deterministic operators and appear to be robust in noisy environments.
- In evaluating a population of  $n$  strings, the GA implicitly estimates the average fitnesses of all schemas that are present in the population, and increasing or decreasing their representation. This simultaneous implicit evaluation of large number of schemas in a population of  $n$  strings is known as *implicit parallelism*. This ability makes them less susceptible to local maxima and noise.

The traditional representation used by the genetic algorithms community is the binary scheme according to which a chromosome is a string the form  $\langle b_1, b_2, \dots, b_m \rangle$ , where  $b_1, b_2, \dots, b_m$  are called **alleles** (either zeros or ones). Since the binary alphabet offers the maximum number of schemata per bit of information of any coding [145], its use has become very popular among scientists. This



1	0	1	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Representation of the number 35.5072 using  
binary encoding

3	5	5	0	7	2
---	---	---	---	---	---

Representation of the number 35.5072 using  
floating point encoding

Figure 3.3: Representing the same number using binary and floating point encodings.

coding also facilitates theoretical analysis of the technique and allows elegant genetic operators. However, since the “implicit parallelism” property of GAs does not depend on using bit strings [148] it is worthwhile to experiment with larger alphabets, and even with new genetic operators. In particular, for optimization problems in which the parameters to be adjusted are continuous, a floating point representation scheme seems a logical choice. According to this representation, a chromosome is a string of the form  $\langle d_1, d_2, \dots, d_m \rangle$ , where  $d_1, d_2, \dots, d_m$  are digits (numbers between zero and nine). Consider the examples shown in Figure 3.3, in which the same value is represented using binary and floating point encoding.

The term “floating” may seem misleading since the position of the implied decimal point is at a fixed position, and the term “fixed point representation” seems more appropriate. However, the reason that the term “floating point” is preferred is because in this representation each variable (representing a parameter to be optimized) may have the point at any position along the string. This means that even when the point is fixed for each gene, is not necessarily fixed along the chromosome. Therefore, some variables could have a precision of 3 decimal places, while others are integers, and still they could all be represented with the same

string. Nevertheless, the term **real-coded** GAs is also used in the literature [158] [159].

Floating point representation is faster and easier to implement, and provides a higher precision than its binary counterpart, particularly in large domains, where binary strings would be prohibitively long. One of the advantages of floating point representation is that it has the property that two points close to each other in the representation space must also be close in the problem space, and vice versa [148]. This is not generally true in the binary approach, where the distance in a representation is normally defined by the number of different bit positions.

Goldberg [158] has presented a theory of convergence for real-coded or floating-point GAs, and also real numbers and other alphabets have been proposed [159], particularly for numerical optimization, in a resemblance of the power of evolutionary strategies [160] in this domain. As Eshelman and Schaffer [161] point out, a lot of researchers in the GA community have agreed to use real-coded genetic algorithms for numerical optimization despite of the fact that there are theoretical arguments that seem to show that small alphabets should be more effective than large alphabets. Practitioners, on the other hand, have shown that real-coded genes work better in practice [162]. A few attempts have been made to develop a theoretical defense of this representation scheme, from which the recent work by Eshelman and Schaffer deserves special attention [161]. One of the main abilities of real-coded GAs is their capacity to exploit the gradualness of functions of continuous variables (where gradualness is taken to mean that small changes in the variables correspond to small changes in the function) [161] [159].

## 3.2 Multiobjective Optimization using GAs

Goldberg [145] indicates that the notion of genetic search in a multicriteria problem dates back to the late 60s, in which Rosenberg's [163] study contained a suggestion that would have led to multicriteria optimization if he had carried it out as presented. His suggestion was to use multiple **properties** (nearness to some specified chemical composition) in his simulation of the genetics and chemistry of a population of single-celled organisms. Since his actual implementation contained only one single property, the multiobjective approach could not be shown in his work, but it was a starting point for researchers interested in this topic.

Genetic algorithms require scalar fitness information to work, which means that when approaching multicriteria problems, we need to perform a scalarization of the objective vectors. One problem is that it is not always possible to derive a global criterion based on the formulation of the problem. In the absence of information, objectives tend to be given equivalent importance, and when we have some understanding of the problem, we can combine them according to the information available, probably assigning more importance to some objectives. Optimizing a combination of the objectives has the advantage of producing a single compromise solution, requiring no further interaction with the decision maker [164]. The problem is, that if the optimal solution cannot be accepted, either because the function used excluded aspects of the problem which were unknown prior to optimization or because we chose an inappropriate setting of the coefficients of the combining function, additional runs may be required until a suitable solution is found.

### 3.3 Use of aggregating functions

Several attempts have been made to combine the objective functions in different ways, as Fonseca and Fleming report [164]. One general approach involves the use of aggregating functions. Several attempts are reported in the literature, and will be described next.

#### 3.3.1 Weighted sum approach

Jakob et al. [165] assign weights that estimate the importance of each objective. The problem with this approach is precisely how to determine such weights when we do not have enough information about the problem. In this case, the optimal point obtained will be a function of the coefficients used to combine the objectives. We normally use a simple linear combination of the objectives and we can generate the trade-off surface (the term “trade-off” in this context refers to the fact that we are trading a value of one objective function for a value of another function or other functions) by varying the weights. This approach is very simple and easy to implement, but it has the disadvantage of missing concave portions of the trade-off curve [166].

#### 3.3.2 Reduction to a single objective

Ritzel and Wayland [166] suggest to code the GA in such a way that all the objectives, except for one, are constant (constrained to a single value), and the remaining objective becomes the fitness function for the GA. Then, through a process of running the GA numerous times with different values of the constrained objectives, a trade-off surface can be developed. The obvious drawback of this approach is that it is time-consuming, and the coding of the objective functions may be difficult or even impossible for certain problems.

### 3.3.3 Goal attainment

Wilson and Macleod [167] used this method to solve an optimization problem. In this method, a vector of weights relating the relative under- or over-attainment of the desired goals must be elicited from the decision maker in addition to the goal vector. By varying the weights, we can generate the set of noninferior solutions, even for nonconvex problems [141]. In the case of underattainment of the desired goals, a smaller weighting coefficient is associated with a more important objective. For overattainment of the desired goals, a smaller weighting coefficient is associated with a less important objective [52].

### 3.3.4 Use of Penalty Functions

The basic idea of this approach is to “punish” the fitness value of a chromosome whenever the solution produced violates some of the constraints imposed by the problem. Theoretically, the penalty decreases when the value of the penalty function coefficient is increased and convergence is achieved by increasing the penalty function coefficient to infinity [168]. However, a large value for the penalty function coefficient causes ill conditioning in the optimization process and results in numerical instability or slow convergence. Furthermore, since the value of the penalty function coefficient is unknown, much experimentation is required to find an appropriate value. The augmented Lagrangian method has been suggested by Adeli and Cheng [168] to deal with this problem. They integrate the penalty function method with the primal-dual method, which is based on sequential minimization of the Lagrangian function. Instead of only a single penalty function coefficient, in this approach two parameters associated with each constraint are used, and there is no need that the Lagrangian multipliers go to infinity to ensure

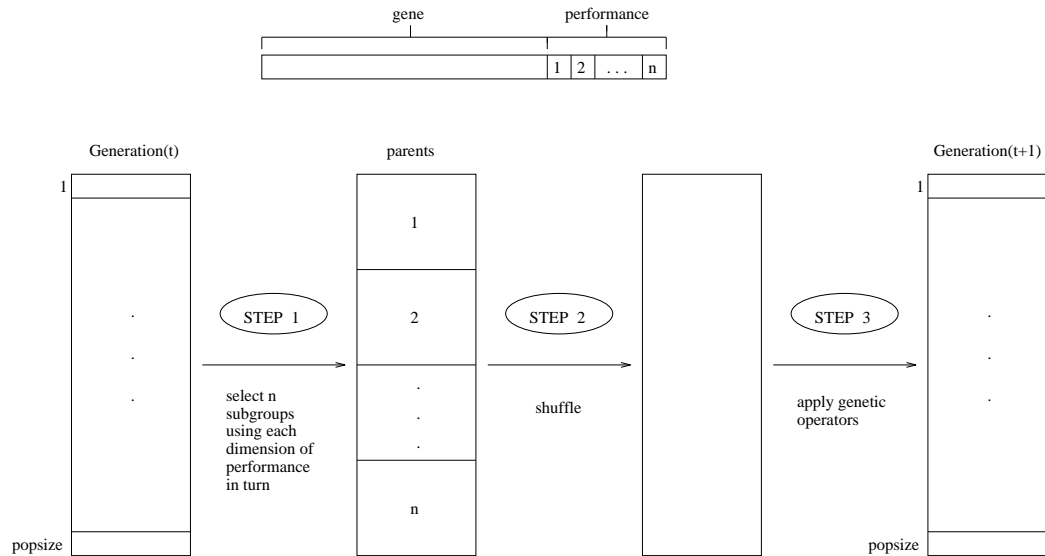


Figure 3.4: Schematic of VEGA selection.

convergence. Nevertheless, the problem that remains is that penalty functions are generally problem dependent, and therefore difficult to establish.

## 3.4 Non-Pareto approaches

To overcome the difficulties involved in the aggregating approach, much work has been devoted to the development of alternative approaches based on ranking [169]. We will examine next some of the most popular **non-Pareto approaches**.

### 3.4.1 VEGA

David Schaffer [170] extended Grefenstette's GENESIS program [171] to include multiple objective functions. Schaffer's approach was to use an extension of the Simple Genetic Algorithm (SGA) that he called the Vector Evaluated Genetic Algorithm (VEGA), and that differed of the first only in the way in which selection was performed. This operator was modified so that at each generation

a number of sub-populations was generated by performing proportional selection according to each objective function in turn. Thus, for a problem with  $k$  objectives,  $k$  sub-populations of size  $N/k$  each would be generated, assuming a total population size of  $N$ . These sub-populations would be shuffled together to obtain a new population of size  $N$ , on which the GA would apply the crossover and mutation operators in the usual way. This process is illustrated in Figure 3.4 (taken from Schaffer [170]). Schaffer realized that the solutions generated by his system were non-inferior in a local sense, because their non-inferiority is limited to the current population, and while a locally dominated individual is also globally dominated, the converse is not necessarily true [170]. An individual who is not dominated in one generation may become dominated by an individual who emerges in a later generation. Also, he noted that the so-called “speciation” problem could arise from his approach (i.e., we could have the evolution of “species” within the population which excel on different aspects of performance). This problem arises because this technique selects individuals who excel in one dimension of performance, without looking at the other dimensions. The potential danger doing that is that we could have individuals with “middling” performance in all dimensions, which could be very useful for compromise solutions, but that will not survive under this selection scheme, since they are not in the extreme for any dimension of performance (i.e., they do not produce the best value for any objective function, but only moderately good values for all of them). Speciation is undesirable because it is opposed to our goal of finding a compromise solution. Schaffer suggested some heuristics to deal with this problem. For example, to use a heuristic selection preference approach for non-dominated individuals in each generation, to protect our “middling” chromosomes. Also, crossbreeding among the

“species” could be encouraged by adding some mate selection heuristics instead of using the random mate selection of the traditional GA.

Although Schaffer reported some success, Richardson et al. [172] noted that the shuffling and merging of all the sub-populations corresponds to averaging the fitness components associated with each of the objectives. Since Schaffer used proportional fitness assignment, these were in turn proportional to the objectives themselves [164]. Therefore, the resulting expected fitness corresponded to a linear combination of the objectives where the weights depended on the distribution of the population at each generation [172]. As a consequence, different non-dominated individuals were generally assigned different fitness values. This problem becomes more severe when we have a concave trade-off surface because points in concave regions of the trade-off surface cannot be found by optimizing a linear combination of the objectives, no matter what set of weights we use.

### 3.4.2 Lexicographic ordering

The basic idea of this technique is that the designer ranks the objectives in order of importance. The optimum solution is then found by minimizing the objective functions, starting with the most important one and proceeding according to the order of importance of the objectives [52]. Fourman [173] suggested a selection scheme based on lexicographic ordering. In a first version of his algorithm, objectives were assigned different priorities by the user and each pair of individuals were compared according to the objective with the highest priority. If this resulted in a tie, the objective with the second highest priority was used, and so on. A second version of this algorithm, reported to work surprisingly well, consisted of randomly selecting the objective to be used in each comparison. As in VEGA, this corresponds to averaging fitness across fitness components, each



component being weighted by the probability of each objective being chosen to decide each tournament [164]. However, the use of pairwise comparisons makes an important difference with respect to VEGA, since in this case scale information is ignored. Therefore, the population may be able to see as convex a concave trade-off surface, depending on its current distribution, and on the problem itself.

### 3.4.3 Evolutionary Strategies

Kursawe [174] formulated a multiobjective version of evolutionary strategies [160] (ESs). Selection consisted of as many steps as objective functions had the problem. At each step, one of these objectives was selected randomly according to a probability vector, and used to delete a fraction of the current population. After selection, the survivors became the parents of the next generation. The map of the trade-off surface was produced from the points evaluated during the run. Since the environment was allowed to change over time, diploid individuals were necessary to keep recessive information stored.

### 3.4.4 Weighted Sum

Hajela and Lin [175] included the weights of each objective in the chromosome, and promoted their diversity in the population through fitness sharing. Their goal was to be able to simultaneously generate a family of Pareto optimal designs corresponding to different weighting coefficients in a single run of the GA. Besides using sharing, Hajela and Lin used a vector evaluated approach based on VEGA to achieve their goal.

## 3.5 Pareto-based approaches

We will also review some of the main **Pareto-based approaches**.

### 3.5.1 Pareto-based fitness assignment

This approach was first proposed by Goldberg [145] to solve the problems of Schaffer's approach. He suggested the use of non-domination ranking and selection to move a population toward the Pareto front in a multiobjective problem. The basic idea is to find the set of strings in the population that are Pareto non-dominated by the rest of the population. These strings are then assigned the highest rank and eliminated from further contention. Another set of Pareto nondominated strings are determined from the remaining population and are assigned the next highest rank. This process continues until the population is suitably ranked. Goldberg also suggested the use of some kind of niching to keep the GA from converging to a single point on the front. A niching mechanism such as sharing [176] would allow the GA to maintain individuals all along the non-dominated frontier. Hilliard et al. [177] used a Pareto optimality ranking method to handle the objectives of minimizing cost and minimizing delay in a scheduling problem. They tentatively concluded that the Pareto optimality ranking method outperformed the VEGA method. The Pareto method was found to be superior to a VEGA by Liepins et al. [178] when applied to a variety of set covering problems. Ritzel et al. [166] also used non-dominated ranking and selection combined with deterministic crowding [179] as the niching mechanism. They applied the GA to a groundwater pollution containment problem in which cost and reliability were the objectives. Though the actual Pareto front was unknown, Ritzel et al. used the best trade-off surface found by a domain-specific algorithm, called MICCP (Mixed Integer Chance Constrained Programming), to compare

the performance of the GA. They found that selection according to Pareto non-domination was superior to both VEGA and non-domination with deterministic crowding, at least for finding points near or on the front found by MICCP.

### 3.5.2 Multiple Objective Genetic Algorithm

Fonseca and Fleming [180] have proposed a scheme in which the rank of a certain individual corresponds to the number of chromosomes in the current population by which it is dominated. Consider, for example, an individual  $x_i$  at generation  $t$ , which is dominated by  $p_i^{(t)}$  individuals in the current generation. Its current position in the individuals' rank can be given by [180]:

$$rank(x_i, t) = 1 + p_i^{(t)} \quad (3.2)$$

All non-dominated individuals are assigned rank 1, while dominated ones are penalized according to the population density of the corresponding region of the trade-off surface.

Fitness assignment is performed in the following way [180]:

1. Sort population according to rank.
2. Assign fitness to individuals by interpolating from the best (rank 1) to the worst (rank  $n^* \leq N$ ) in the way proposed by Goldberg [145], according to some function, usually linear, but not necessarily.
3. Average the fitnesses of individuals with the same rank, so that all of them will be sampled at the same rate. This procedure keeps the global population fitness constant while maintaining appropriate selective pressure, as defined by the function used.

As Goldberg and Deb [150] point out, this type of blocked fitness assignment is likely to produce a large selection pressure that might produce premature convergence. To avoid that, Fonseca and Fleming use a niche-formation method to distribute the population over the Pareto-optimal region, but instead of performing sharing on the parameter values, they have used sharing on objective function values [181]. This maintains diversity in the objective function values, but may not maintain diversity in the parameter set, which is an important issue for a decision maker. Furthermore, this approach may not be able to find multiple solutions in problems where different Pareto-optimal points correspond to the same objective function value.

In this approach, it is possible to evolve only a certain region of the trade-off surface, by combining Pareto dominance with partial preference information in the form of a goal vector. While the basic ranking scheme remains unaltered, as we perform a Pareto comparison of the individuals, then those objectives which already satisfy their goals will not be selected. If we specify fully unattainable goals, then objectives will never be excluded from comparison. Changing the goal values during the search alters the fitness landscape accordingly and allows the decision maker to magnify a particular region of the trade-off surface.

### 3.5.3 Non-dominated Sorting Genetic Algorithm

The Non-dominated Sorting Genetic Algorithm (NSGA) was proposed by Srinivas and Deb [182], and is based on several layers of classifications of the individuals. Before the selection is performed, the population is ranked on the basis of nondomination: all nondominated individuals are classified into one category (with a dummy fitness value, which is proportional to the population size, to provide an equal reproductive potential for these individuals). To maintain

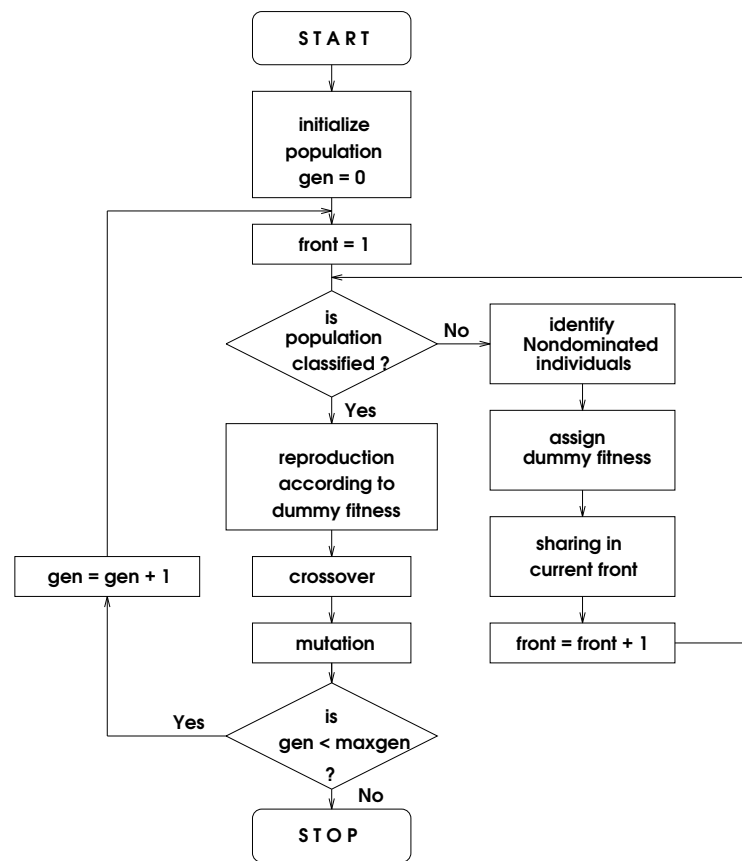


Figure 3.5: Flowchart of the Nondominated Sorting Genetic Algorithm (NSGA).

the diversity of the population, these classified individuals are shared with their dummy fitness values. Then this group of classified individuals is ignored and another layer of nondominated individuals is considered. The process continues until all individuals in the population are classified. A stochastic remainder proportionate selection was used for this approach. 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 nondominated regions, and results in quick convergence of the population toward such regions. Sharing, by its part, helps to distribute it over this region. The efficiency of NSGA lies in the way multiple objectives are reduced to a dummy fitness function using a nondominated sorting procedure. With this approach, any number of objectives can be solved [181], and both maximization and minimization problems can be handled. Figure 3.5 (taken from Srinivas and Deb [181]) shows the general flow chart of this approach.

### 3.5.4 Niched Pareto GA

Horn and Nafpliotis [183] proposed a tournament selection scheme based on Pareto dominance. Instead of limiting the comparison to two individuals, a number of other individuals in the population was used to help determine dominance. When both competitors were either dominated or non-dominated (i.e., there was a tie), the result of the tournament was decided through fitness sharing [176]. Population sizes considerably larger than usual were used so that the noise of the selection method could be tolerated by the emerging niches in the population [164].

The pseudocode for Pareto domination tournaments assuming that all of the objectives are to be maximized is presented below [183].  $S$  is an array of the

$N$  individuals in the current population, *random\_pop\_index* is an array holding the  $N$  indices of  $S$ , in a random order, and  $t_{dom}$  is the size of the comparison set.

**function** selection /\* Returns an individual from the current population  $S$  \*/

**begin**

    shuffle(random\_pop\_index); /\* Re-randomize random index array \*/

    candidate\_1 = random\_pop\_index[1];

    candidate\_2 = random\_pop\_index[2];

    candidate\_1\_dominated = **false**;

    candidate\_2\_dominated = **false**;

**for** comparison\_set\_index = 3 to  $t_{dom} + 3$  **do**

        /\* Select  $t_{dom}$  individuals randomly from  $S$  \*/

**begin**

            comparison\_individual = random\_pop\_index[comparison\_set\_index];

            if  $S[\text{comparison\_individual}]$  dominates  $S[\text{candidate\_1}]$

**then** candidate\_1\_dominated = **true**;

            if  $S[\text{comparison\_individual}]$  dominates  $S[\text{candidate\_2}]$

**then** candidate\_2\_dominated = **true**;

**end** /\* end for loop \*/

**if** ( candidate\_1\_dominated **AND**  $\neg$  candidate\_2\_dominated )

**then** return candidate\_2;

**else if** (  $\neg$  candidate\_1\_dominated **AND** candidate\_2\_dominated )

**then** return candidate\_1;

**else**

        do sharing;

**end**

Scaling of the objectives determines the convexity of the trade-off surface, so that if we use a non-linear rescaling, the objective values may convert a concave surface into a convex one, and vice-versa. Pareto-ranking is blind to the convexity of the trade-off surface, but this does not mean that it always precludes speciation [164], since this can still occur if certain regions of the trade-off region are simply easier to find than others. However, Pareto-ranking eliminates sensitivity to the possible non-convexity of the trade-off surface, and also it encourages the production of compromise solutions.

It should be noticed that even when Pareto-based ranking correctly assigns all non-dominated individuals the same fitness, does not guarantee that the Pareto set be uniformly sampled, since finite populations will tend to converge to only one optimum when several equivalent optima are present, due to stochastic errors in the selection process [164]. This phenomenon, which is known as **genetic drift**, has been observed in both natural and artificial evolution, and can also occur in Pareto-based GA optimization [164].

Goldberg and Richardson [176] proposed the use of fitness sharing to prevent genetic drift and to promote the sampling of the whole Pareto set by the population. Fonseca and Fleming [180] implemented fitness sharing in the objective domain and provided theory for estimating the necessary niche sizes based on the properties of the Pareto set. Horn and Nafpliotis [183] also arrived at a form of fitness sharing in the objective domain, and suggested the use of a metric combining both the objective and the decision variable domains, leading to what they called **nested sharing**.

Another interesting aspect that has been considered in GA-based multiobjective optimization has been the viability of crossover. This is an important issue,



because we could have different genetic representations across different regions of the trade-off surface, and therefore we could need to restrict crossover to happen only locally [145]. So far, crossover restrictions have been implemented based on the distance between individuals in the objective domain, either directly [180] or indirectly [175].

## 3.6 Summary of Methods

In this section I will summarize the contents of this chapter, providing at the same time some additional information regarding the mathematical programming techniques and the Operations Research concepts that inspired them.

### 1. Use of aggregating functions

- (a) Weighted sum: This method is based on the **Weighting objectives method** introduced in Chapter 2. The ideal vector does not have to be known, but a set of weights and probably some scaling factor have to be provided by the user. This method will normally produce only the min-max optimum, but not the Pareto front or the Pareto front unless a lot of weight combinations are tried.
- (b) Reduction to a single objective: The central idea of this technique is based on the  $\varepsilon$ —**constrained method** introduced in Chapter 2. The ideal vector is not required, but in some cases it is very difficult to transform the objectives into constants. This method will normally produce only the min-max optimum, but not the Pareto optimum or the Pareto front.
- (c) Goal-attainment: This method is based on the **Global criterion** and **Goal Programming** techniques introduced in Chapter 2. The ideal

vector or another target vector must be provided by the user. This method will normally produce only the min-max optimum, but not the Pareto optimum or the Pareto front.

- (d) Use of Penalty functions: This method is based on the  $\varepsilon$ -**constrained method** and the **Weighting objectives method** introduced in Chapter 2. The ideal vector is not necessary, but the assignment of penalty values is not always a trivial task. This method will normally produce only the min-max optimum, but not the Pareto optimum or the Pareto front.

## 2. Non-Pareto approaches

- (a) VEGA: This method is somehow based on **Multiattribute utility theory**, although within the context of genetic algorithms. The ideal vector is not required, and the Pareto front can be generated. However, since this approach really selects individuals based on their capabilities in a single dimension, in problems with highly conflicting objectives the good trade-offs will be eliminated. Both the min-max optimum and the Pareto optimum may be found in certain problems.
- (b) Lexicographic ordering: This method is based on the **Lexicographic method** introduced in Chapter 2. The ideal vector is not required. Normally, only the min-max optimum may be found using this approach.
- (c) Evolutionary strategies: The method developed by Kursawe [174] is also based on the **Lexicographic method**, although in this case, a fraction of the population is replaced, keeping a historical record of the best individuals found. The ideal vector is not required, but normally only the min-max optimum will be found using this approach.

- (d) Weighted sum: This method is based on the **Weighting objectives method**, but in this case sharing is used to avoid convergence to a single solution. Also, a hybrid approach in which a weighted sum is combined with VEGA has been proposed by Hajela and Lin [175].

### 3. Pareto-based approaches

- (a) Pareto-based fitness assignment: This method is based somehow on the **Sequential multiobjective problem solving method (SEMOPS)** introduced in Chapter 2. The ideal vector is not required, but an algorithm to determine non-dominance is necessary. Therefore, the Kuhn-Tucker conditions have to be incorporated within the selection strategy of the genetic algorithm.
- (b) MOGA: This is a variation of the previous method in which the fitness of each individual is assigned based on the number of chromosomes that dominate it. Again, the Kuhn-Tucker conditions have to be incorporated within the selection strategy of the genetic algorithm to determine non-dominance, but the ideal vector does not have to be provided.
- (c) NSGA: This method is also a variation of Pareto-based fitness assignment, although in this case the individuals are assigned dummy fitness values. The ideal vector does not have to be known, but we need to implement an algorithm to check for non-dominance (the Kuhn-Tucker conditions can be used for that sake). The ideal vector does not have to be provided by the user.

(d) NPGA: This method is based on **Multiattribute utility theory**.

The ideal vector does not have to be provided, but the sharing factor and the tournament size are critical for the good performance of this method.

## Chapter 4

# Implementation of MOSES

In this chapter, I will describe the capabilities, limitations, and implementation details of MOSES (Multiobjective Optimization of Systems in the Engineering Sciences), a system developed to compare several multiobjective optimization techniques applied to different engineering design optimization problems. MOSES encompasses several programs and modules that allow the use of mathematical programming techniques and GA-based approaches to solve multiobjective optimization problems. All the code was developed in C (compiled with the GNU C compiler), and run on a Sun Sparc Workstation. Parts of it were translated from original FORTRAN implementations found in Osyczka [2]. I will start by talking about some of the main algorithms used by MOSES, and then I will describe two random search methods that can be used to find Pareto optimal solutions, together with the min-max optimum. After that, I will describe some of the main features of the Interactive Multicriterion Optimization System implemented by Osyczka, which was translated to C and incorporated into MOSES. Finally, I will talk about the different GA-based approaches implemented.

The first step in developing a mathematical programming technique to deal with multiobjective optimization problems is to be able to identify, given a set of feasible solutions to the problem, which of them are Pareto optimal. After that,

it will be desirable to agree upon a concept of optimality in this context, in case the designer desires a single final solution. For the scope of this work, we will adopt the concept of min-max optimum for that sake. Therefore, we will provide also an algorithm that will give us the min-max optimum from a certain set of solutions.

## 4.1 Generating Pareto Optimal Solutions

We have seen already the concept of Pareto optimality, but I haven't described so far, any algorithm that can identify the Pareto optimal solutions from a given set of feasible solutions. Osyczka [2] provides an algorithm that is based on the contact theorem, which is one of the main theorems in multiobjective optimization [184].

First, let us define a negative cone [2]. The negative cone in  $R^k$  is the set

$$C^- = \{\bar{f} \in R^k | \bar{f} \leq 0\} \quad (4.1)$$

Thus, the contact theorem is:

A vector  $f^*$  is a Pareto optimal solution for the general multiobjective optimization problem if and only if

$$(C^- + \bar{f}^*) \cap F = \{f^*\} \quad (4.2)$$

A graphical illustration of this theorem for a two criterion problem is shown in Figure 4.1 (taken from Osyczka [2]).

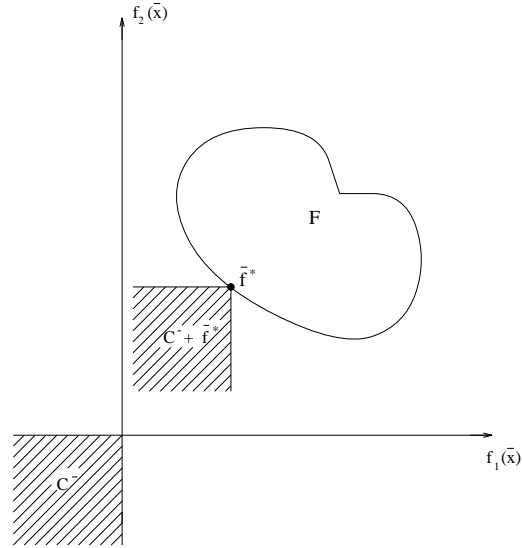


Figure 4.1: Graphical illustration of the contact theorem.

Consider two solutions  $\bar{x}^{(1)}$  and  $\bar{x}^{(2)}$  for which we may have two specific cases

$$(1)(C^- + \bar{f}(\bar{x}^{(1)})) \subset (C^- + \bar{f}(\bar{x}^{(2)})) \quad (4.3)$$

$$(2)(C^- + \bar{f}(\bar{x}^{(1)})) \supset (C^- + \bar{f}(\bar{x}^{(2)})) \quad (4.4)$$

A graphical illustration of these cases is presented in Figure 4.2 (taken from Osyczka [2]).

We denote  $\bar{x}^{(l)} = [x_1^l, x_2^l, \dots, x_n^l]^T =$  any given point in  $X$ ,

$f(\bar{x}^{(l)}) = [f_1(\bar{x}^{(l)}), f_2(\bar{x}^{(l)}), \dots, f_k(\bar{x}^{(l)})]^T =$  vector of objective functions for the point  $\bar{x}^{(l)}$ ,

$\bar{x}_j^p = [x_{1j}^p, x_{2j}^p, \dots, x_{nj}^p]^T =$  The  $j$ th Pareto optimal solution,

$\bar{f}_j^p = [f_{1j}^p, f_{2j}^p, \dots, f_{kj}^p]^T =$  vector of objective functions for the  $j$ th Pareto optimal solution.

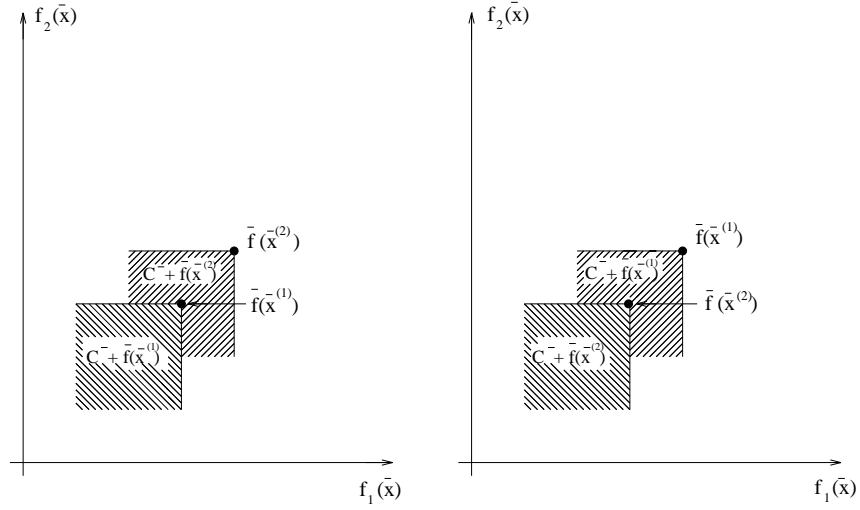


Figure 4.2: Graphical illustration of equations (4.3) and (4.4).

Now the problem is to choose from any given set of solutions

$L = \{1, 2, \dots, l, \dots, l^a\}$ , the set of Pareto optimal solutions

$J = \{1, 2, \dots, j, \dots, j^a\}$ .

The main idea behind the Pareto algorithm is the following. Let  $\bar{x}^{(l)}$  be a new solution to be considered. If in the set of Pareto optimal solutions there is a solution  $x_j^p$  such that it

- (i) satisfies (4.3) then  $\bar{x}^{(l)}$  is substituted for  $\bar{x}_j^p$ , or
- (ii) satisfies (4.4) then  $\bar{x}^{(l)}$  is discarded.

If none of the solutions from the Pareto set satisfies either 4.3 or 4.4, then  $\bar{x}^{(l)}$  becomes a new Pareto optimal solution.

The steps of the algorithm are the following [2]:

- (1) Read  $k$  (number of objective functions),  $n$  (number of decision variables),  $l^a$  (number of solutions available).
- (2) Set  $f_{i1}^p = \infty$  for  $i = 1, 2, \dots, k$  and  $j^a = 1$ .
- (3) Set  $l = 1$ .
- (4) Read  $\bar{x}^{(l)}$  and  $f(\bar{x}^{(l)})$ .



- (5) Set  $j = 1$ .
- (6) If for every  $i \in I$  we have  $f_i(\bar{x}^{(l)}) < f_{ij}^p$  then substitute  $\bar{x}_j^p = \bar{x}^{(l)}$  and  $f_j^p = f(\bar{x}^{(l)})$  and go to 10, otherwise go to 7.
- (7) If for every  $i \in I$  we have  $f_i(\bar{x}^{(l)}) > f_{ij}^p$  then go to 10 otherwise go to 8.
- (8) Set  $j = j + 1$ .
- (9) If  $j > j^a$  then  $j^a = j^a + 1$  and  $x_{ja}^p = \bar{x}^{(l)}$  and  $\bar{f}_j^p = \bar{f}(\bar{x}^{(l)})$  and go to 10, otherwise go to 6.
- (10) Set  $l = l + 1$ .
- (11) If  $l \leq l^a$ , then go to 4, otherwise go to 12.
- (12) Print  $\bar{x}_j^p$  and  $\bar{f}_j^p$  for  $j = 1, 2, \dots, j^a$ .

This algorithm is called PARETO by Osyczka [2] and it was translated from FORTRAN to C and incorporated into MOSES.

## 4.2 The min-max algorithm

This algorithm chooses from any given set of solutions  $L = \{1, 2, \dots, l, \dots, l^a\}$ , the min-max optimal solution as defined in Chapter 1. We assume that the ideal vector  $\bar{f}^0$  is given.

The steps of the algorithm are the following [2]:

- (1) Read  $k$  (number of objective functions),  $n$  (number of decision variables),  $l^a$  (number of available solutions),  $\bar{f}^0$  (ideal vector).
- (2) Set  $v_1^* = \infty$ .
- (3) Set  $l = 1$ .
- (4) Read  $\bar{x}^{(l)}$  and  $\bar{f}(\bar{x}^{(l)})$ .
- (5) Evaluate vector  $\bar{z}(\bar{x}^{(l)})$  using formula (1.27).
- (6) If  $\bar{z}(\bar{x}^{(l)}) = 0$  then retain this solution as the optimum since there is no better solution, and go to 11, otherwise go to 7.

(7) Find the maximal values of all the steps of formula (1.29) for the points  $\bar{x}^{(l)}$ . These values are denoted  $v_r$  for  $r = 1, 2, \dots, k$ , and can be evaluated as follows

$$v_1 = \max_{i \in I} \{z_i(x^{(l)})\} \quad (4.5)$$

and then  $I_1 = \{i_1\}$ , where  $i_1$  is the index for which the value of  $z_i(\bar{x}^{(l)})$  is maximal,

$$v_2 = \max_{i \in I, i \notin I_1} \{z_i(\bar{x}^{(l)})\} \quad (4.6)$$

and the  $I_2 = \{i_1, i_2\}$ , where  $i_2$  is the index for which the value of  $z_i(\bar{x}^{(l)})$  is maximal,

$$v_r = \max_{i \in I, i \notin I_{r-1}} \{z_i(\bar{x}^{(l)})\} \quad (4.7)$$

and then  $I_r = \{I_{r-1}, i_r\}$ , where  $i_r$  is the index for which the value of  $z_i(\bar{x}^{(l)})$  is maximal,

$$v_k = z_i(\bar{x}^{(l)}) \quad \text{for } i \in I \text{ and } i \notin I_{k-1} \quad (4.8)$$

(8) Replace  $v_r^*$  by  $v_r$  for  $r = 1, 2, \dots, k$  and retain this solution as the optimum if the following function is satisfied

$$v_1 < v_1^* \vee_{r \in \{2, \dots, k\}} ((v_r < v_r^*) \wedge_{s \in \{1, \dots, r\}} (v_s = v_s^*)) \quad (4.9)$$

where  $v_1^*, v_2^*, \dots, v_k^*$  is the set of optimal values of relative increments ordered non-increasingly.

(9) Set  $l = l + 1$ .

(10) If  $l \leq l^a$  then go to 4, otherwise go to 11.

(11) Print  $x^*, l^*, \bar{f}(\bar{x}^*), \bar{z}(\bar{x}^*)$ .

This algorithm is called MINMAX by Osyczka [2] and it was also translated from FORTRAN to C and incorporated into MOSES, both in the mathematical programming software and in some of the GA-based approaches.

### 4.3 Monte Carlo Methods

I also implemented the two Monte Carlo methods used by Osyczka [2] to find the min-max optimum. These methods are called **exploratory** because a point is generated by means of a rule which disregards the results previously obtained. In particular, the Monte Carlo method picks up a certain number of points at random over the estimated range of all the variables of the problem. This is done formally by obtaining the randomly selected value for  $x_i$  from the following formula

$$x_i = x_i^a + \delta_i(x_i^b - x_i^a) \quad \text{for } i = 1, 2, \dots, n \quad (4.10)$$

where  $x_i^a$  is the estimated or given lower limit for  $x_i$ ,  $x_i^b$  is the estimated or given upper limit for  $x_i$ , and  $\delta_i$  is a random number between zero and one. I employed the same random number generator used by the genetic algorithm to implement the FORTRAN function RANF of the original program.

If we want to generate the values of variables for  $l^a$  points, we start by generating random numbers  $\delta_i$  for each point, and then use equation (4.10) to obtain the values of the variables  $x_i$ . After that, we test each generated point for violation and discard it if it is not a feasible solution. If the point is in the feasible region, we evaluate the objective function for that point. The best result is taken

as the minimum, and a new set of random numbers is generated for each of  $l^a$  points.

The two Monte Carlo methods described by Osyczka [2] to find the min-max optimum are presented next.

### 4.3.1 Monte Carlo method 1

In this method, the space of variables is explored twice, first searching for the ideal vector  $\bar{f}^0$  and then searching for the min-max optimum. The algorithm is the following [2]:

Do steps 1, 2, 3, 4, for  $l = 1, 2, \dots, l^a$

- (1) Generate a random point  $\bar{x}^{(l)}$ .
- (2) If the point  $\bar{x}^{(l)}$  is not in the feasible region go to 1, otherwise go to 3.
- (3) Evaluate  $f_i(\bar{x}^{(l)})$  for  $i = 1, 2, \dots, k$ .
- (4) Replace  $f_i^0$  by  $f_i(\bar{x}^{(l)})$  for every  $i$  for which  $f_i(\bar{x}^{(l)}) < f_i^0$ .

Do steps 5, 6, 7, 8, for  $l = 1, 2, \dots, l^a$

- (5) Generate a random point  $\bar{x}^{(l)}$ .
- (6) If the point  $\bar{x}^{(l)}$  is not in the feasible region go to 5, otherwise go to 7.
- (7) Evaluate  $f_i(\bar{x}^{(l)})$  for  $i = 1, 2, \dots, k$ .
- (8) Call MINMAX to check if the point  $\bar{x}^{(l)}$  is the min-max optimum.

### 4.3.2 Monte Carlo method 2

Here, the space of variables is explored only once, and the Pareto set is generated while searching for the ideal vector  $\bar{f}^0$ . Then, this set is analyzed to check which solution is the min-max optimum. The algorithm is the following [2]:

Do steps 1, 2, 3, 4, 5, for  $l = 1, 2, \dots, l^a$

- (1) Generate a random point  $\bar{x}^{(l)}$ .

(2) If the point  $\bar{x}^{(l)}$  is not in the feasible region go to 1, otherwise to 3.

(3) Evaluate  $f_i(\bar{x}^{(l)})$  for  $i = 1, 2, \dots, k$ .

(4) Replace  $f_i^0$  by  $f_i(\bar{x}^{(l)})$  for every  $i$  for which  $f_i(\bar{x}^{(l)}) < f_i^0$ .

(5) Call PARETO for checking if the point  $\bar{x}^{(l)}$  is Pareto optimal.

Do steps 6, 7 for  $j = 1, 2, \dots, j^a$

(6) Evaluate  $f_i(\bar{x}_j^p)$  for  $i = 1, 2, \dots, k$ .

(7) Call MINMAX for checking if the point  $\bar{x}_j^p$  is the min-max optimum.

There are several trade-offs between these two methods. For example, the second method uses less CPU time than the first, because the space of variables is explored only once, but it also requires much more memory since the whole Pareto set has to be stored. Obviously, the designer normally wants to analyze the entire Pareto set in order to take a decision, but as I mentioned before, this set could be too large and the computational resources available could be insufficient for that sake. Osyczka recommends the reduction of this set by introducing constraints of the form

$$f_i(\bar{x}) \leq f_i^0 \text{ for } i = 1, 2, \dots, k$$

where values of  $f_i^0$  are chosen by the designer.

The second method should be preferred for problems with a large number of constraints and for discrete programming problems, because in those cases we expect to have a small Pareto set. The main advantage of exploratory methods in general is their flexibility, since they can be applied both to linear and non-linear programming problems. However, they are normally recommended only for cases where a few decision variables are handled because otherwise, they could take too long to find a reasonable good solution.

## 4.4 Osyczka's Multicriterion Optimization System

This system was developed at the Technical University of Cracow, and its FORTRAN implementation is provided in Osyczka's book [2]. A C translation of that code was incorporated into MOSES, and its contents are explained next.

Osyczka's system contains several multiobjective optimization methods:

(1) Min-max method : Equation (1.27) is used to determine the elements of the vector  $\bar{z}(\bar{x})$  (see Chapter 1).

(2) Global criterion method : Equation (2.12) is used as the global function (see Chapter 2).

(3) Weighting min-max method : This is a combination of the weighting method and the min-max approach that can find the Pareto set of solutions for both convex and non-convex problems. The equation

$$\bigwedge_{i \in I} (z_i(\bar{x}) = \max\{w_i z_i'(\bar{x}), w_i z_i''(\bar{x})\}) \quad (4.11)$$

is used to determine the elements of vector  $\bar{z}(\bar{x})$ .

(4) Pure weighting method : Equation (2.5) is used to determine a preferred solution.

(5) Normalized weighting method :  $\bar{f}(\bar{x})$  is used in equation (2.5).

Since all these methods require the ideal vector, the user is given the choice of providing it, or letting the system to find it automatically. For this purpose, the system includes two single criterion optimization techniques:

(i) The flexible tolerance (FT) method : Is a sequential method in which a point is established on the basis of the previously obtained results. Based on this information, the method will know where the minimum is likely to be so that the

appropriate search direction may be established. Normally sequential methods, even when are more efficient and more highly developed than exploratory methods, tend to be designed to solve only continuous convex problems. However, this particular method can deal with non-linear models [185]. A detailed explanation of this algorithm is provided in Chapter 6.

(ii) The direct and random search (DRS) method : It is a mixture of an exploratory and a sequential method. The direct search method [186] starts from the point chosen by the user and seeks a minimum. Then, a new starting point is generated at random and then the direct search method seeks a better solution. The procedure is repeated  $n$  times, and each time the direct search method starts from a new point where the value of  $n$  is given by the user. The best result from all searches is taken as the minimum.

## 4.5 Implementing GA-based approaches

Several Multiobjective Optimization approaches based on genetic algorithms were implemented and incorporated into MOSES. The main body of the system was based on the Simple Genetic Algorithm (SGA) originally implemented by Goldberg [145] and then translated to C by R. E. Smith and modified by Jeff Erickson. However, this C implementation had to be modified to support both binary and floating point representations. The first, follows the traditional scheme presented by Goldberg in his book [145], and the second corresponds to the representation of integer or real numbers by using each allele as a digit, and placing the decimal point according to the lower and upper bounds of the design variables.

MOSES has an automatic encoding facility. The user can choose among three different types of variables:

(1) Integer : No decimals are considered. The user has to provide a lower and an upper limit, and MOSES determines the number of genes that each chromosome needs to represent it, depending on the representation scheme selected (either binary or floating point). To compute the actual size of the chromosome, I used the expression:

$$size = (int)\{\log_m(upper\ bound - lower\ bound) + 0.9\} \quad (4.12)$$

where  $m = 2$  for binary representation, and  $m = 10$  for floating point representation.

(2) Discrete : A list of either integers or reals will be provided directly by the user. Discrete variables are always considered reals by MOSES. The user has to provide the size of the list, and the corresponding value of each one of them. To compute the actual size of the chromosome, the following expression is used:

$$size = (int)\{\log_m(num\ vals) + 0.9\} \quad (4.13)$$

where  $m = 2$  for binary representation, and  $m = 10$  for floating point representation.

(3) Real : A range and a number of digits of precision are asked of the user. To compute the actual size of the chromosome, the lower and upper bound are recomputed based on the precision considered, using the expression:

$$new\ bound = previous\ bound * (10^{precision}) \quad (4.14)$$

where *previous bound* is the corresponding lower or upper bound provided by the user and *precision* is an integer that indicates the number of digits to be



used after the decimal point. With these new bounds, equation (4.12) is used to compute the chromosome size.

MOSES expects all the input from a file, and it needs another file to generate its output. The parameters of the GA (maximum number of generations, maximum number of runs, population size, crossover rate, mutation rate, maximum number of generations and random numbers seed) can be passed on the command line, but suitable defaults are included in the program.

Two-point crossover is always used, and even when tournament selection was used for all the experiments of this thesis (except when indicated), roulette wheel selection and stochastic remainder selection are also available.

The system was designed in a modular fashion, so that the user only has to plug the particular decode, report and fitness modules to start working. Everything else remains normally the same, except for the code used for selection, which can be changed according to the user needs. To make things easier, however, the code of each method was incorporated as a single unit in a separate program, and compiled as an isolated entity. The techniques incorporated into MOSES are briefly explained in the following subsections, indicating in each case what were the changes performed to the standard code.

#### 4.5.1 Lexicographic Method

This is a variation of the lexicographic method [187] mentioned by Fonseca and Fleming [188] in which individuals are compared in pairs (binary tournament selection) and the individual with the highest fitness according to any of the objective functions (the corresponding objective function is randomly chosen) is selected.

### 4.5.2 Schaffer's VEGA

In this approach, developed by Schaffer [170], the population is randomly divided into  $m$  subgroups, where

$$m = \frac{\text{population size}}{\text{number of objectives}} \quad (4.15)$$

The individuals in each subgroup are selected according to a single objective function, and then crossover is performed over the entire population. This can be easily implemented by using a flag that indicates, for each subgroup, which is the objective function to use to determine the fitness of each individual.

### 4.5.3 Hajela's Approach

Hajela and Lin [175] proposed the use of a utility function of the form:

$$\bar{U} = \sum_{i=1}^l W_i \frac{F_i}{F_i^*} \quad (4.16)$$

where  $F_i^*$  are the scaling parameters for the objective criterion,  $l$  is the number of objective functions, and  $W_i$  are the weighting factors for each objective function  $F_i$ . In my implementation, I used a min-max approach to determine the utility function, so that the scaling factor was the ideal vector.

Hajela's approach also uses a sharing function of the form:

$$\phi(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{sh}}\right)^\alpha, & d_{ij} < \sigma_{sh} \\ 0, & \text{otherwise} \end{cases} \quad (4.17)$$

where  $\alpha = 1$  for this work,  $d_{ij}$  is a metric indicative of the distance between designs  $i$  and  $j$ , and  $\sigma_{sh}$  is the sharing parameter, which is typically chosen between 0.01 and 0.1. The fitness of a design  $i$  is then modified as:

$$f_{s_i} = \frac{f_i}{\sum_{j=1}^M \phi(d_{ij})} \quad (4.18)$$

where  $M$  is the number of designs located in vicinity of the  $i$ -th design.

Hajela incorporates weight combinations into the chromosomic string, so our implementation was extended to accommodate the additional genes required, according to the number of weight combinations provided by the user. Under Hajela's representation, a single number represents not the weight itself, but a combination of weights. For example, the number 4 (under floating point representation) could represent the vector  $X_w = (0.4, 0.6)$  for a problem with two objective functions. Then, sharing is done on the weights.

Finally, a mating restriction mechanism was imposed, to avoid members within a radius  $\sigma_{mat}$  to cross. The value of  $\sigma_{mat} = 0.15$  used by Hajela was adopted in my implementation.

#### 4.5.4 The Niche Pareto Genetic Algorithm

In this method, only the function **select()** is redefined, as indicated in [183] and shown in Chapter 3. The values of  $t_{dom}$  and  $\sigma_{share}$  should be provided by the user. Equivalence class sharing [183] is done on the attribute values (i.e., on the vector of objective function values), and it was implemented according to the following algorithm [183]:

```
function selection
```

```
begin
```

```
    :
```

```
        else if nichecount[candidate_1] > nichecount[candidate_2]
```

```
            then return candidate_2;
```

```

        else return candidate_1;
    end
end

```

The value of *nichecount* is generated by the equivalence class sharing algorithm. The idea is that the best individual will be determined to be the one that has the least number of individuals in its niche and thus the smallest niche count.

#### 4.5.5 The Nondominated Sorting Genetic Algorithm

This is a variant of Goldberg's suggestion [145] for using a nondominated ranking procedure to select the best individuals in the population. The algorithm is shown in Figure 3.5, and its explanation may be found in [181]. In this case, there is a pre-processing stage before starting the actual selection process. This algorithm was implemented by looping during as many fronts as objective functions the problem has. At each step of the loop, we have to determine if an individual dominates in as many objectives as  $m - k$  where  $m$  is the maximum number of objectives, and  $k$  is the current front (starting by zero). So, at the first front, the absolute nondominated individuals will be selected, and as the number of front increases, individuals which dominate only partially will be chosen. The idea of this method is to assign the highest dummy fitness values to the first front, and decrease such dummy fitness value as we increase the number of fronts. Sharing is done on these dummy fitness values, to spread diversity and avoid convergence towards a single solution. The value of  $\sigma_{share}$  has to be provided by the user. Since the checking for nondominance implies an  $O(N^2)$  algorithm, a temporary array of  $n \times k$  dimensions was created ( $n$ =population size, and  $k$ =number of objectives) to hold the solution vectors of each chromosome in the population, avoiding a repeating function evaluation at each step of the nested loop.

It should be noticed that this implementation uses stochastic remainder proportionate selection [148] as indicated by Srinivas and Deb [181].

#### 4.5.6 The Multiple Objective Genetic Algorithm

This method is another variant of Goldberg's suggestion [145] proposed by Fonseca and Fleming [180] for using nondominated ranking to select individuals. The idea is to assign to each individual a rank based on the number of individuals that dominate it plus one. According to this scheme, nondominated individuals are assigned a rank of 1 (nobody dominates them). The fitness value of each individual is computed according to the following expression:

$$fitness_i = \frac{\text{population size}}{\text{rank}_i} \quad (4.19)$$

This expression guarantees that the nondominated individuals always get the highest fitness values.

Instead of averaging the fitness of the individuals that have the same rank, as suggested by Fonseca and Fleming [180], I just implemented a form of sharing on these fitness values, in which the value of  $\sigma_{share}$  is determined according to the following expression:

$$\sigma_{share} = A/N \quad (4.20)$$

where  $N$ =population size, and

$$A = \sum_{i=1}^k \prod_{j=1, j \leq i}^k (M_j - m_j) \quad (4.21)$$

Here,  $k$ =number of objective functions,  $M_j$  is the maximum value that the  $j$ th objective function takes within the  $N$  individuals under consideration,

and  $m_j$  is the minimum value that the  $j$ th objective function takes within the  $N$  individuals of the population being considered.

#### 4.5.7 An Approach Based on a Weighted Min-Max Strategy

This is really a variant of Hajela's idea, in which a few changes were introduced by me:

1. The initial population is generated in such a way that all their individuals constitute feasible solutions. This can be ensured by checking that none of the constraints is violated by the solution vector encoded by the corresponding chromosome.
2. The user should provide a vector of weights, which are used to spawn as many processes as weight combinations are provided (normally this number will be reasonably small). Each process is really a separate genetic algorithm in which the given weight combination is used in conjunction with a min-max approach to generate a single solution. Notice that in this case the weights do not have to be encoded in the chromosome as in Hajela's approach.
3. After the  $n$  processes are terminated ( $n$ =number of weight combinations provided by the user), a final file is generated containing the Pareto set, which is formed by picking up the best solution from each of the processes spawned in the previous step.
4. Since this approach requires knowing the ideal vector, the user is given the choice to provide such values directly (in case he/she knows them) or to use another genetic algorithm to generate it. This additional program works in a similar manner, spawning  $k$  processes ( $k$ =number of objective functions),

where each process corresponds to a genetic algorithm responsible for a single objective function. When all the processes terminate, there will be a file containing the ideal vector, which turns out to be simply the best values produced by each one of the spawned processes.

5. The crossover and mutation operators were modified to ensure that they produced only feasible solutions. Whenever a child encodes an infeasible solution, it is replaced by one of its parents.
6. Notice that the Pareto solutions produced by this method are guaranteed to be feasible, as opposed to the other GA-based methods in which there could be convergence towards a non-feasible solution.

#### **4.5.8 An Approach Based on Min-Max Selection with Sharing**

This is another approach that I tried, in which a Min-Max selection strategy replaces the Pareto ranking selection scheme previously reported in the literature, and sharing is used to avoid the GA converging to a single solution. The basic algorithm is the following:

1. The initial population is generated as in the previous approach, ensuring that all the individuals at generation zero encode only valid solutions.
2. By exploring the population at each generation, the local ideal vector is produced. This is done by comparing the values of each objective function in the entire population.

3. The binary tournament selection algorithm is modified, so that instead of comparing the fitnesses of two individuals, we compare their maximal deviations with respect to the local ideal vector. If one dominates the other, then it wins the tournament, and if there is a tie, then sharing is used to decide who is the winner, in a way similar to the NPGA. This means that we count the number of individuals within the niche of each one of the competitors, and the individual with a lower count wins.
4. The crossover and mutation operators were modified as in the previous algorithm to ensure that they produced only feasible solutions. Whenever a child encodes an infeasible solution, it is replaced by one of its parents.
5. Notice that the Pareto solutions produced by this method are also guaranteed to be feasible, as opposed to the other GA-based methods in which there could be convergence towards an infeasible solution.

#### 4.5.9 The GA optimizer for single-objective problems

Using the GA itself as an optimizer for single-objective problems is a controversial topic, mainly because the difficulties found to adjust its parameters (i.e., population size, maximum number of generations, mutation and crossover rate) [189]. Since one of the goals of this work is to be able to produce a reliable design optimization system, this is a natural problem to face. In practice, GA parameters are empirically adjusted in a trial and error process that could take quite a long time in some cases.

For several months, I experimented with a very simple methodology, explained below, for a variety of engineering design optimization problems. The results that I obtained led me to think that it was a reasonable choice to use in MOSES. The method is the following:



- Choose a certain value for the random number seed and make it a constant.
- Make constants for the population size and the maximum number of generations (I normally use 100 chromosomes and 50 generations, respectively).
- Loop the mutation and crossover rates from 0.1 to 0.9 at increments of 0.1 (this is actually a nested loop). This implies that 81 runs are necessary. In each step of the loop, the population is not reinitialized.
- For each run, update 2 files. One contains only the final costs, and the other has a summary that includes, besides the cost, the corresponding values of the design parameters and the mutation and crossover rates used.
- When the whole process ends, the file with the costs is sorted in ascending order, and the smallest value is searched for in the other file, returning the corresponding design parameters as the final answer.

So far, I have found much better results using floating point representation with this methodology, and I think that will be the trend shown in the problems solved in this work. This approach is actually a dynamic adjustment of parameters, because the population is initialized only once in the process, so that the individuals' fitness continues improving while changing the crossover and mutation rates. Notice that even when we could know the crossover and mutation rates produced the best answer, running the GA once with those parameters will not necessary generate the exact same answer. The reason is that the population at the moment of finding the best result could have been recombined and improved several times, being quite different of the random initial population of a simple GA. This procedure has some resemblance with Eshelman's CHC Adaptive Search Algorithm [190], but in my case I do not use any re-feeding of the population through high mutation values when it has stabilized, nor a highly disruptive

recombinator operator that produces offsprings that are maximally different from both parents. My approach uses a conventional two-point crossover and it exhibits its best behavior with a floating point representation in numerical optimization problems.

## Chapter 5

# Some Engineering Design Examples

The following set of engineering design optimization problems was chosen from the literature, in order to test the different techniques included in MOSES:

1. Design of an I-beam
2. Machining recommendations
3. Design of a machine tool spindle
4. Design of a 10-bar plane truss
5. Design of a 25-bar space truss
6. Design of a 200-bar plane truss
7. Design of a robot arm
8. Design of a combinatorial circuit

This chapter is devoted to the complete descriptions of such problems, and because of their length, I will not include any further material here. Therefore the results of their analysis will be presented in the next chapter.

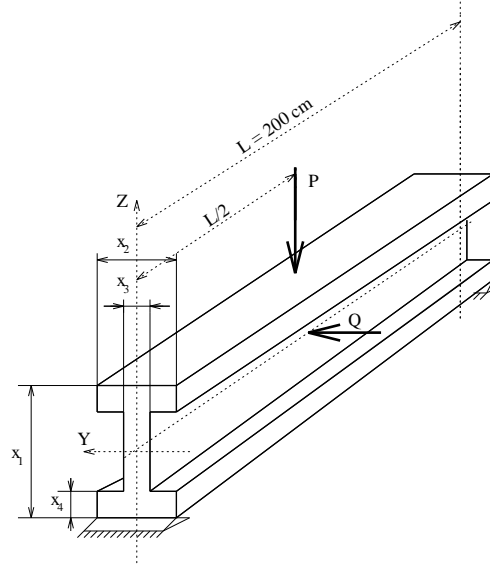


Figure 5.1: The simply supported I-beam of Example 1.

## 5.1 Example 1 : Design of an I-beam

The multiobjective optimization problem is formulated as follows [1]:

Find the dimensions of the beam presented in Figure 5.1 (taken from Osyczka [1]) which satisfy the geometric and strength constraints and which optimize the following criteria:

1. cross section area of the beam which for the give length reflects its volume; and
2. static deflection of the beam for the displacement under the force P.

Both criteria are to be minimized. It should be noted that these criteria are contrary to one another (i.e., the best solution for the first objective function gives the worst solution for the second one and viceversa).

It is assumed that:

1. Permissible bending stress of the beam material  $k_g = 16 \text{ kN/cm}^2$ .
2. Young's Modulus of Elasticity  $E = 2 \times 10^4 \text{ kN/cm}^2$ .
3. Maximal bending forces  $P = 600 \text{ kN}$  and  $Q = 50 \text{ kN}$ .

The vector of the decision variables is  $x = [x_1, x_2, x_3, x_4]^T$ . Their values will be given in centimeters. The geometric constraints are:

$$10 \leq x_1 \leq 80, \quad 10 \leq x_2 \leq 50, \quad 0.9 \leq x_3 \leq 5, \quad 0.9 \leq x_4 \leq 5 \quad (5.1)$$

The strength constraint is

$$\frac{M_y}{W_y} + \frac{M_z}{W_z} \leq k_g \quad (5.2)$$

where  $M_y$  and  $M_z$  are maximal bending moments in  $Y$  and  $Z$  directions respectively;  $W_y$  and  $W_z$  are section moduli in  $Y$  and  $Z$  directions respectively. For the forces acting the values of  $M_y$  and  $M_z$  are  $30,000 \text{ kN-cm}$  and  $2,500 \text{ kN-cm}$  respectively. The section moduli can be expressed as follows:

$$W_y = \frac{x_3(x_1 - 2x_4)^3 + 2x_2x_4[4x_4^2 + 3x_1(x_1 - 2x_4)]}{6x_1} \quad (5.3)$$

$$W_z = \frac{(x_1 - 2x_4)x_3^3 + 2x_4x_2^3}{6x_2} \quad (5.4)$$

Thus the strength constraint is:

$$16 - \frac{180,000x_1}{x_3(x_1 - 2x_4)^3 + 2x_2x_4[4x_4^2 + 3x_1(x_1 - 2x_4)]} - \frac{15,000x_2}{(x_1 - 2x_4)x_3^3 + 2x_4x_2^3} \geq 0 \quad (5.5)$$

The objective functions can be expressed as follows

1. Cross-section area

$$f_1(x) = 2x_2x_4 + x_3(x_1 - 2x_4) \text{ cm}^2 \quad (5.6)$$

2. Static deflection

$$f_2(x) = \frac{Pl^3}{48EI} \text{ cm} \quad (5.7)$$

where  $I$  is the moment of inertia which can be calculated from

$$I = \frac{x_3(x_1 - 2x_4)^3 + 2x_2x_4[4x_4^2 + 3x_1(x_1 - 2x_4)]}{12} \quad (5.8)$$

After substitution the second objective function is

$$f_2(x) = \frac{60,000}{x_3(x_1 - 2x_4)^3 + 2x_2x_4[4x_4^2 + 3x_1(x_1 - 2x_4)]} \quad (5.9)$$

Test No.	$v$ (sfm)	$f$ (ipr)	$d$ (in)	SR ( $\mu$ in)	SI (% undamaged)	TL (min)	MRR (in <sup>3</sup> /min)
1	625	0.002	0.050	25	24	150.6	0.75
2	625	0.010	0.050	150	31	108.1	3.75
3	625	0.018	0.050	230	19	89.8	6.75
4	625	0.010	0.097	86	29	80.2	7.28
5	625	0.018	0.097	180	20	29.4	13.10
6	966	0.005	0.078	30	55	32.7	4.52
7	966	0.015	0.078	210	45	24.2	13.56
8	1200	0.010	0.050	95	30	30.5	7.20

Table 5.1: Machinability data for 390 die cast/carbonide/wet.

## 5.2 Example 2 : Machining recommendations

The problem is the following [191]:

Machinability tests on 390 die cast aluminum cut with VC-3 carbide cutting tools were conducted over the following ranges of speed, feed rates, and depths of cut:

Cutting speed ( $v$ ) :      600 sfm to 1200 sfm  
 Feed rate ( $f$ ):              0.002 ipr to 0.018 ipr  
 Depth of cut ( $d$ ):          0.050 in to 0.100 in

Table 5.1 (taken from Ghiassi et al. [191]) provides the cutting conditions and associated machining performance measures (criteria). The data in this table were used to develop first-order predicting equations for the performance variables SR, SI, TL, and MRR in terms of the controllable variables  $v$ ,  $f$  and  $d$ , in logarithmic transformed coordinates. The following equations represent the least-squares fit to the data; the feed and depth of cut have been multiplied by 1000 to ensure that their logarithms are positive.

$$\begin{aligned}
\ln SR &= 7.49 - 0.44 \ln v + 1.16 \ln(1000f) - 0.61 \ln(1000d) \\
\ln SI &= -4.13 + 0.92 \ln v - 0.16 \ln(1000f) + 0.43 \ln(1000d) \\
\ln TL &= 21.90 - 1.94 \ln v - 0.30 \ln(1000f) - 1.04 \ln(1000d) \\
\ln MRR &= -11.33 + \ln v + \ln(1000f) + \ln(1000d)
\end{aligned} \tag{5.10}$$

Bounds on the values of the controllable variables are defined below to reflect the ranges over which the machinability tests were run, and bounds on the values of the performance variables.

$$\begin{aligned}
600 &\leq v \leq 1200 \text{ } sfm \\
0.002 &\leq f \leq 0.018 \text{ } ipr \\
0.05 &\leq d \leq 0.10 \text{ } in
\end{aligned} \tag{5.11}$$

$$\begin{aligned}
SR &\leq 75 \text{ } \mu in \\
SI &\geq 50\% \text{ } undamaged \\
TL &\geq 30 \text{ } min
\end{aligned} \tag{5.12}$$

In order to express the variables in Equations (5.11) and (5.12) in the same form as in the performance model, Equation (5.10), their logarithmic transformations are given in Equations (5.13) and (5.14):

$$\begin{aligned}
6.3969 &\leq \ln v \leq 7.0901 \\
0.6931 &\leq \ln(1000f) \leq 2.8904 \\
3.9120 &\leq \ln(1000d) \leq 4.6052
\end{aligned} \tag{5.13}$$



$$\begin{aligned}
-0.44 \ln v + 1.16 \ln(1000f) - 0.61 \ln(1000d) &\leq -3.17 \\
-0.92 \ln v + 0.16 \ln(1000f) - 0.43 \ln(1000d) &\leq -8.04 \\
1.94 \ln v + 0.30 \ln(1000f) + 1.04 \ln(1000d) &\leq 18.50
\end{aligned} \tag{5.14}$$

The natural logarithms of  $SR$ ,  $TL$  and  $SI$  have been substituted from Equation (5.10) to obtain Equation (5.14). The constraint of  $TL$  was multiplied by  $-1$  to maintain the same direction of inequality as the other constraints.

To simplify the statement of the problem, let  $x_1 = \ln v$ ,  $x_2 = \ln(1000f)$ ,  $x_3 = \ln(1000d)$ , and the vector  $x = (x_1, x_2, x_3)$ . Also, define  $z_1(x) = \ln SR$ ,  $z_2(x) = \ln SI$ ,  $z_3(x) = \ln TL$ , and  $z_4(x) = \ln MRR$ . Again, for simplicity, we will refer to the objective functions as  $z_1$ ,  $z_2$ ,  $z_3$  and  $z_4$ , respectively, and define the vector  $z = (-z_1, z_2, z_3, z_4)$ . Note that minimizing  $z_1 = \ln SR$  is equivalent to maximizing  $-z_1$ . Accordingly, the components of  $z$  [from Equation (1.7)] may be expressed as:

$$\begin{aligned}
-z_1 &= -7.49 + 0.44x_1 - 1.16x_2 + 0.61x_3 \\
z_2 &= -4.13 + 0.92x_1 - 0.16x_2 + 0.43x_3 \\
z_3 &= 21.90 - 1.94x_1 - 0.30x_2 - 1.04x_3 \\
z_4 &= -11.331 + x_1 + x_2 + x_3
\end{aligned} \tag{5.15}$$

Maximize  $z$ , as defined by Equation (5.15), under the following constraints:

$$\begin{aligned}
6.3969 &\leq x_1 \leq 7.0901 \\
0.6931 &\leq x_2 \leq 2.8904 \\
3.9120 &\leq x_3 \leq 4.6052
\end{aligned} \tag{5.16}$$

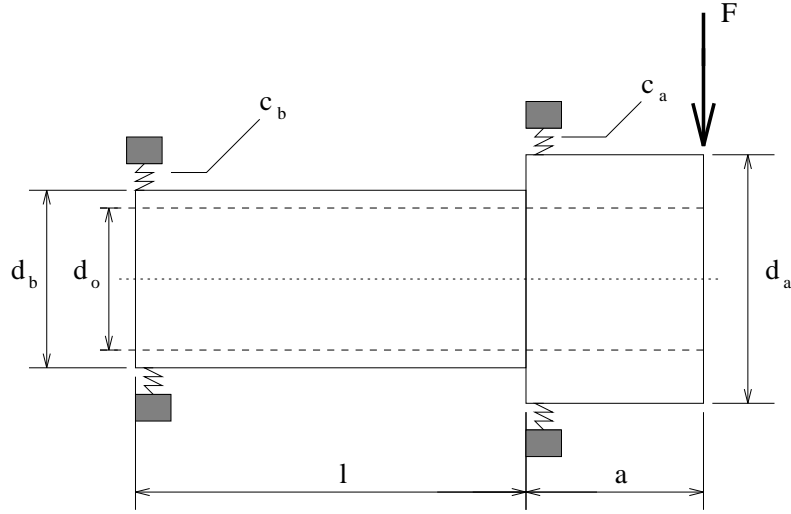


Figure 5.2: Sketch of the machine tool spindle used for Example 3.

$$\begin{aligned}
 -0.44x_1 + 1.16x_2 - 0.61x_3 &\leq -3.1725 \\
 -0.92x_1 + 0.16x_2 - 0.43x_3 &\leq -8.0420 \\
 1.94x_1 - 0.30x_2 - 1.04x_3 &\leq 18.4988
 \end{aligned} \tag{5.17}$$

Inequalities (5.16) and (5.17) correspond to inequalities (5.13) and (5.14), respectively.

### 5.3 Example 3 : Design of a machine tool spindle

Consider the problem of a preliminary design of a machine tool spindle as presented in Figure 5.2 (taken from Eschenauer et al. [192]). The formulation of the multiobjective optimization problem is to minimize  $f_1(x)$  and  $f_2(x)$  as defined below [192].

Objectives:

Volume of the spindle

$$f_1(x) = \frac{\pi}{4} [a(d_a^2 - d_o^2) + l(d_b^2 - d_o^2)] \quad (5.18)$$

Static displacement under the force F

$$f_2(x) = \frac{Fa^3}{3EI_a} \left(1 + \frac{l}{a} \frac{I_a}{I_b}\right) + \frac{F}{c_a} \left[ \left(1 + \frac{a}{l}\right)^2 + \frac{c_a a^2}{c_b l^2} \right] \quad (5.19)$$

where  $I_a$  and  $I_b$  are the moments of inertia

$$I_a = 0.049(d_a^4 - d_o^4), \quad I_b = 0.049(d_b^4 - d_o^4) \quad (5.20)$$

$c_a$  and  $c_b$  are bearing stiffnesses

$$c_a = 35400 |\delta_{ra}|^{\frac{1}{9}} d_a^{\frac{10}{9}}, \quad c_b = 35400 |\delta_{rb}|^{\frac{1}{9}} d_b^{\frac{10}{9}} \quad (5.21)$$

with  $\delta_{ra}$  and  $\delta_{rb}$  as preloads of the bearings.

Constraints:

a) side constraints (bounds)

$$\begin{aligned} g_1(x) &= l - l_g \leq 0 \\ g_2(x) &= l_k - l \leq 0 \end{aligned} \quad (5.22)$$

$$g_3(x) = d_{a1} - d_a \leq 0$$

$$g_4(x) = d_a - d_{a2} \leq 0$$

$$g_5(x) = d_{b1} - d_b \leq 0$$

$$g_6(x) = d_b - d_{b2} \leq 0$$

$$g_7(x) = d_{om} - d_o \leq 0$$

(5.23)

b) Designer's proportion requirements

$$\begin{aligned} g_8(x) &= p_1 d_o - d_b \leq 0 \\ g_9(x) &= p_2 d_b - d_a \leq 0 \end{aligned} \quad (5.24)$$

c) Maximal radial runout of the spindle nose  $\Delta$

$$g_{10}(x) = |\Delta_a + (\Delta_a - \Delta_b) \frac{a}{l}| - \Delta \leq 0 \quad (5.25)$$

where  $\Delta_a$  and  $\Delta_b$  are the radial runouts of the front and the end bearings.

For this example, it is assumed that  $d_a$  must be chosen from the set  $X_3 = \{80, 85, 90, 95\}$ , and  $d_b$  from the set  $X_4 = \{75, 80, 85, 90\}$ . Additionally, the following constant parameters are assumed:

$d_{om}=25.00$ mm	$d_{a1}=80.00$ mm
$d_{a2}=95.00$ mm	$d_{b1}=75.00$ mm
$d_{b2}=90.00$ mm	$p_1=1.25$
$p_2=1.05$	$l_k=150.00$ mm
$l_g=200.00$ mm	$a=80.00$ mm
$E = 210,000.0$ N/mm <sup>2</sup>	$F = 10,000$ N
$\Delta_a = 0.00540000$ mm	$\Delta_b = -0.00540000$ mm
$\Delta = 0.01000000$ mm	$\delta_{r_a} = -0.00100000$ mm
$\delta_{r_b} = -0.00100000$ mm	

The decision variables are  $l$ ,  $d_o$ ,  $d_a$  and  $d_b$ .

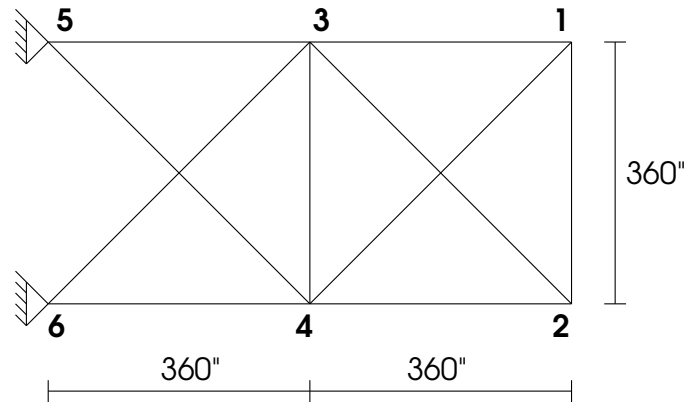


Figure 5.3: 10-bar plane truss used for Example No. 4.

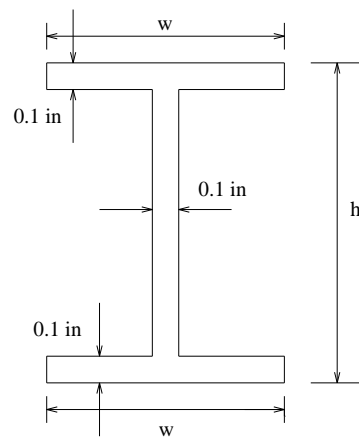


Figure 5.4: Cross-section used for Example No. 4.

## 5.4 Example 4 : Design of a 10-bar plane truss

Consider the 10-bar plane truss shown in Figure 5.3 [193]. The problem is to find the cross-sectional area of each member of this truss, such that we minimize its weight, the displacement of each free node, and the stress that each member has to support. The weight of the truss is given by  $f(x)$ .

$$f(x) = \sum_{j=1}^{10} \rho A_j L_j \quad (5.26)$$

where  $x$  is the candidate solution,  $A_j$  is the cross-sectional area of the  $j$ th member,  $L_j$  is the length of the  $j$ th member, and  $\rho$  is the weight density of the material. The assumed data are: modulus of elasticity,  $E = 1.09 \times 10^4$  ksi,  $\rho = 0.10$  lb/in<sup>3</sup>, and a load of 100 kips in the negative y-direction is applied at nodes 2 and 4. The maximum allowable stress of each member is called  $\sigma_a$ , and it is assumed to be  $\pm 25$  ksi. The maximum allowable displacement of each node (horizontal and vertical) is represented by  $u_a$ , and is assumed to be 2 inches. The minimum allowable cross-section area is 0.10 in<sup>2</sup> for all members. The cross-section of each element can be different, and is defined by the I-section shown in Figure 5.4, with the depth and width as design variables. The web thickness and flange thickness are each kept fixed at 0.1 in. The problem has, therefore, 20 design variables.

## 5.5 Example 5 : Design of a 25-bar space truss

Consider the 25-bar space truss taken from Rajeev and Khrisnamoorthy [194] shown in Figure 5.5. The problem is to find the cross-sectional area of each member of this truss, such that we minimize its weight, the displacement of each free node, and the stress that each member has to support.

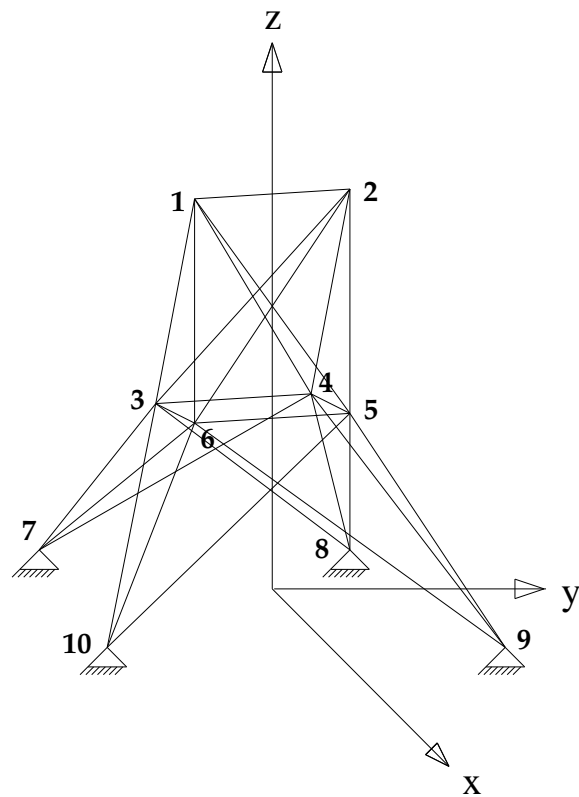


Figure 5.5: 25-bar space truss used for example No. 5.

Node	$F_x$ (lbs)	$F_y$ (lbs)	$F_z$ (lbs)
1	1000	-10000	-10000
2	0	-10000	-10000
3	500	0	0
6	600	0	0

Table 5.2: Loading conditions for the 25-bar space truss shown in Figure 5.5.

Group Number	Members
1	1-2
2	1-4, 2-3, 1-5, 2-6
3	2-5, 2-4, 1-3, 1-6
4	3-6, 4-5
5	3-4, 5-6
6	3-10, 6-7, 4-9, 5-8
7	3-8, 4-7, 6-9, 5-10
8	3-7, 4-8, 5-9, 6-10

Table 5.3: Group membership for the 25-bar space truss shown in Figure 5.5.

Node	X	Y	Z
1	-37.50	0.00	200.00
2	37.50	0.00	200.00
3	-37.50	37.50	100.00
4	37.50	37.50	100.00
5	37.50	-37.50	100.00
6	-37.50	-37.50	100.00
7	-100.00	100.00	0.00
8	100.00	100.00	0.00
9	100.00	-100.00	0.00
10	-100.00	-100.00	0.00

Table 5.4: Coordinates of the joints of the 25-bar space truss shown in Figure 5.5.



Loading conditions are given in Table 5.2, member groupings are given in Table 5.3, and node coordinates are given in Table 5.4. The assumed data are: modulus of elasticity,  $E = 1 \times 10^4$  ksi,  $\rho = 0.10$  lb/in<sup>3</sup>;  $\sigma_a = \pm 40$  ksi,  $u_a = \pm 0.35$  in.

## 5.6 Example 6 : Design of a 200-bar plane truss

Consider the 200-bar plane truss taken from Belegundu [193], shown in Figure 5.6 (taken from Belegundu [193]). The problem is to find the cross-sectional area of each member of this truss, such that we minimize its weight, the displacement of each free node, and the stress that each member has to support.

There are a total of three loading conditions: (1) 1 kip acting in positive x-direction at node points 1, 6, 15, 20, 29, 34, 43, 48, 57, 62, and 71; (2) 10 kips acting in negative y-direction at node points 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 15, 16, 17, 18, 19, 20, 24, 71, 72, 73, 74, and 75; and (3) loading condition 1 and 2 acting together. The 200 elements of this truss linked to 29 groups. The grouping information is shown in Table 5.5. The stress in each element is limited to a value of 10 ksi for both tension and compression members. Young's modulus of elasticity = 30,000 ksi, weight density =  $0.283 \times 10^{-3}$  kips/in<sup>3</sup>.

## 5.7 Example 7 : Design of a robot arm

Consider the PUMA-560 shown in Figure 5.7 (taken from Armstrong et al. [195]). Koski and Osyczka [196] present a multiobjective optimization model of such arm based on its rigid-body dynamics. By using angular coordinates for the PUMA-560 robot, it is possible to calculate the generalized torques at each joint applying the following equation:

Group Number	Member Number
1	1,2,3,4
2	5,8,11,14,17
3	19,20,21,22,23,24
4	18,25,56,63,94,101,132,139,170,177
5	26,29,32,35,38
6	6,7,9,10,12,13,15,16,27,28,30,31,33,34,36,37
7	39,40,41,42
8	43,46,49,52,55
9	57,58,59,60,61,62
10	64,67,70,73,76
11	44,45,47,48,50,51,53,54,65,66,68,69,71,72,74,75
12	77,78,79,80
13	81,84,87,90,93
14	95,96,97,98,99,100
15	102,105,108,111,114
16	82,83,85,86,88,89,91,92,103,104,106,107,109,110,112,113
17	115,116,117,118
18	119,122,125,128,131
19	133,134,135,136,137,138
20	140,143,146,149,152
21	120,121,123,124,126,127,129,130,141,142,144,145,147,148,150,151
22	153,154,155,156
23	157,160,163,166,169
24	171,172,173,174,175,176
25	178,181,184,187,190
26	158,159,161,162,164,165,167,168,179,180,182,183,185,186,188,189
27	191,192,193,194
28	195,197,198,200
29	196,199

Table 5.5: Group membership for the 200-bar plane truss shown in Figure 5.6.

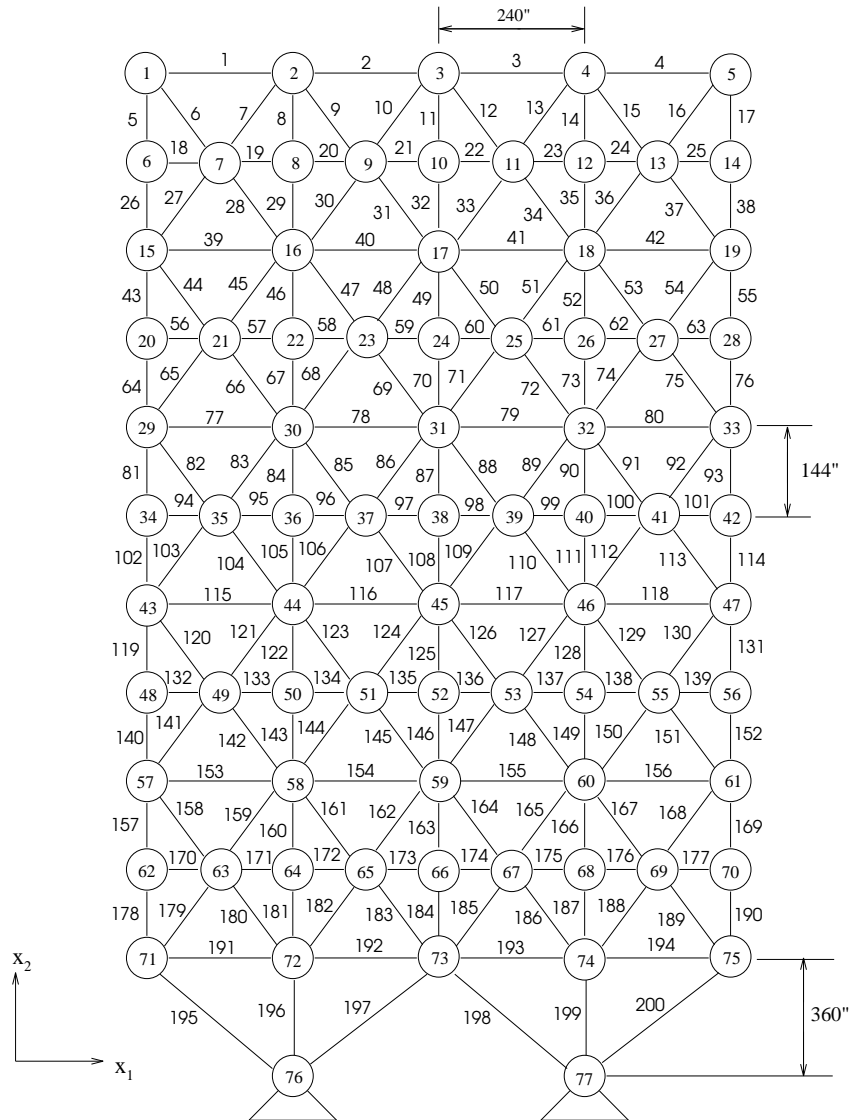


Figure 5.6: 200-bar plane truss used for example No. 6.

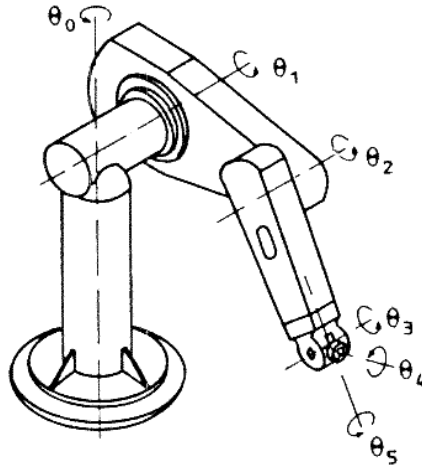


Figure 5.7: PUMA-560 robot arm and schematic representation of coordinate angles  $\theta_i$ .

$$M_{ti} = \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{\theta}_i} \right) - \frac{\partial L}{\partial \theta_i} \quad (5.27)$$

where  $\theta_i$  is the rotation at joint  $i$  and  $\dot{\theta}_i$  is the corresponding angular velocity. The term

$$L = T - V \quad (5.28)$$

represents the Lagrangian function of the mechanical system. Here,  $T$  is the total kinetic energy of the system and  $V$  is the total potential energy. The application of Equation (5.27) to a fully articulated robot arm results in the following nonlinear second-order system of differential equations

$$A\ddot{\theta} + B\dot{\theta}^2 + c - m = 0 \quad (5.29)$$

Here, the vector of angular accelerations is given by

$$\ddot{\theta} = (\ddot{\theta}_1, \ddot{\theta}_2, \dots, \ddot{\theta}_N)^T \quad (5.30)$$

and the vector of squared angular velocities by

$$\begin{aligned} \dot{\theta}^2 = & (\dot{\theta}_1\dot{\theta}_1, \dot{\theta}_1\dot{\theta}_2, \dots, \dot{\theta}_1\dot{\theta}_N | \dot{\theta}_2\dot{\theta}_1, \dot{\theta}_2\dot{\theta}_2, \dots, \dot{\theta}_2\dot{\theta}_N | \dots \dot{\theta}_k\dot{\theta}_1, \\ & \dot{\theta}_k\dot{\theta}_2, \dots, \dot{\theta}_k\dot{\theta}_N | \dots \dot{\theta}_N\dot{\theta}_1, \dot{\theta}_N\dot{\theta}_2, \dots, \dot{\theta}_N\dot{\theta}_N)^T \end{aligned} \quad (5.31)$$

where  $N$  is the number of joints. The corresponding matrices are

$$A = \begin{bmatrix} D_{11} & D_{12} & \dots & D_{1N} \\ D_{21} & D_{22} & \dots & D_{2N} \\ \vdots & & & \\ D_{N1} & D_{N2} & \dots & D_{NN} \end{bmatrix} \quad (5.32)$$

and

$$B = \left[ \begin{array}{cccc|cccc} D_{11}^1 & D_{12}^1 & \dots & D_{1N}^1 & D_{11}^2 & D_{12}^2 & \dots & D_{1N}^2 \\ D_{21}^1 & D_{22}^1 & \dots & D_{2N}^1 & D_{21}^2 & D_{22}^2 & \dots & D_{2N}^2 \\ & \vdots & & & & \vdots & & \\ D_{N1}^1 & D_{N2}^1 & \dots & D_{NN}^1 & D_{N1}^2 & D_{N2}^2 & \dots & D_{NN}^2 \\ & & & & D_{11}^N & D_{12}^N & \dots & D_{1N}^N \\ & & & & D_{21}^N & D_{22}^N & \dots & D_{2N}^N \\ & & & & & \vdots & & \\ & & & & D_{N1}^N & D_{N2}^N & \dots & D_{NN}^N \end{array} \right]$$

where it is assumed that each joint has one degree of freedom. The elements of matrix  $A$  are the inertia terms, and the elements of matrix  $B$  represent the centripetal and Coriolis terms. All these terms depend on the position of the

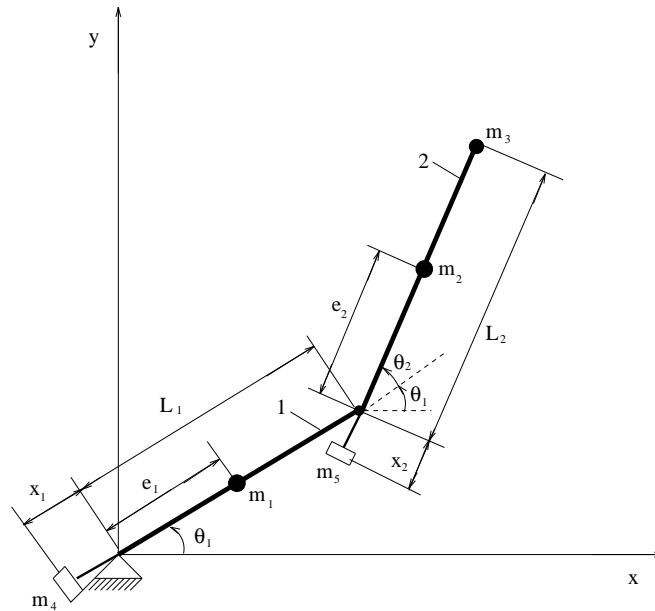


Figure 5.8: Mechanical model of the robot arm used for optimization.

arm—i.e.,  $D_{ij} = D_{ij}(\theta_i)$ —. Vector  $c = (D_1, D_2, \dots, D_N)^T$  includes the gravitational terms  $D_i$  and  $m$  is the vector of torques. Kinetic Equations (5.27), (5.28) and (5.29) represent the rigid-body motion of the arm, and they are geometrically nonlinear because of large rotations of  $\theta_i$ .

The manipulator is an isostatic structure, and thus it is possible to get explicit expressions for all forces and moments in the system. The friction in the joints as well as the flexibility of the arm are not included in the following design model. For the application of optimization methods, a two-member robot arm, which corresponds to the two arms of the PUMA-560 robot in a plane motion, is considered. This arm is assumed to move in the  $xy$ -plane only (corresponding angular coordinates  $\theta_i$  are shown in Figure 5.8 {taken from Koski and Osyczka. [196]}). The masses of the members are  $m_1$  and  $m_2$ . They are located as point masses at distances  $e_1$  and  $e_2$  from the joints. The external load is represented

by the point mass  $m_3$ . In the model used by Koski and Osyczka, only the counterweight masses  $m_4$  and  $m_5$ , as well as their distances from the joints  $x_1$  and  $x_2$  are treated as design variables, whereas all the other quantities are fixed.

The torques of this two-member robot are obtained from the Equation 5.27 and are expressed as follows:

$$\begin{aligned} M_{t1} &= D_{11}\ddot{\theta}_1 + D_{12}\ddot{\theta}_2 + D_{11}^1\ddot{\theta}_1^2 + D_{12}^2\ddot{\theta}_2^2 + (D_{12}^1 + D_{11}^2)\dot{\theta}_1\dot{\theta}_2 + D_1 \\ M_{t2} &= D_{21}\ddot{\theta}_1 + D_{22}\ddot{\theta}_2 + D_{21}^1\ddot{\theta}_1^2 + D_{22}^2\ddot{\theta}_2^2 + (D_{22}^1 + D_{21}^2)\dot{\theta}_1\dot{\theta}_2 + D_2 \end{aligned} \quad (5.33)$$

The coefficients for torque  $M_{t1}$  are:

$$\begin{aligned} D_{11} &= m_1e_1^2 + m_4x_1^2 + m_2e_2^2 + m_3L_2^2 + m_5x_2^2 + (m_2 + m_3 + m_5)L_1^2 \\ &\quad + 2m_2e_2L_1\cos\theta_2 + 2m_3L_1L_2\cos\theta_2 - 2m_5x_2L_1\cos\theta_2 + J_1 + J_2 \end{aligned} \quad (5.34)$$

$$\begin{aligned} D_{12} &= m_2e_2^2 + m_3L_2^2 + m_5x_2^2 + m_2e_2L_1\cos\theta_2 + m_3L_1L_2\cos\theta_2 \\ &\quad - m_5x_2L_1\cos\theta_2 + J_2 \end{aligned} \quad (5.35)$$

$$D_{11}^1 = -m_5x_2L_1[\sin(\theta_0 + \theta_1) + \sin\theta_2] + m_5x_2^2\sin 2\theta_0 \quad (5.36)$$

$$\begin{aligned} D_{12}^2 &= -m_2e_2L_1\sin\theta_2 - m_3L_1L_2\sin\theta_2 - m_5x_2L_1\sin(\theta_0 + \theta_1) \\ &\quad + m_5x_2^2\sin 2\theta_0 \end{aligned} \quad (5.37)$$

$$D_{12}^1 + D_{11}^2 = -2m_2e_2L_1\sin\theta_2 - 2m_3L_1L_2\sin\theta_2 - \\ 2m_5x_2L_1\sin(\theta_0 + \theta_1) + 2m_5x_2^2\sin 2\theta_0 \quad (5.38)$$

$$D_1 = m_1ge_1\cos\theta_1 - m_4gx_1\cos\theta_1 + m_2ge_2\cos\theta_0 + m_3gL_2\cos\theta_0 - \\ m_5gx_2\cos\theta_0 + (m_2 + m_3 + m_5)gL_1\cos\theta_1 \quad (5.39)$$

Coefficients for torque  $M_{t2}$  are:

$$D_{21} = m_2(L_1e_2\cos\theta_2 + e_2^2) + m_3(L_1L_2\cos\theta_2 + L_2^2) - \\ m_5(L_1x_2\cos\theta_2 - x_2^2) + J_2 \quad (5.40)$$

$$D_{22} = m_2e_2^2 + m_3L_2^2 + m_5x_2^2 + J_2 \quad (5.41)$$

$$D_{21}^1 = m_2L_1e_2\sin\theta_2 + m_3L_1L_2\sin\theta_2 + m_5(2x_2^2\sin\theta_0\cos\theta_0 - L_1x_2\sin\theta_2) \quad (5.42)$$

$$D_{22}^2 = 2m_5x_2^2\sin\theta_0\cos\theta_0 \quad (5.43)$$



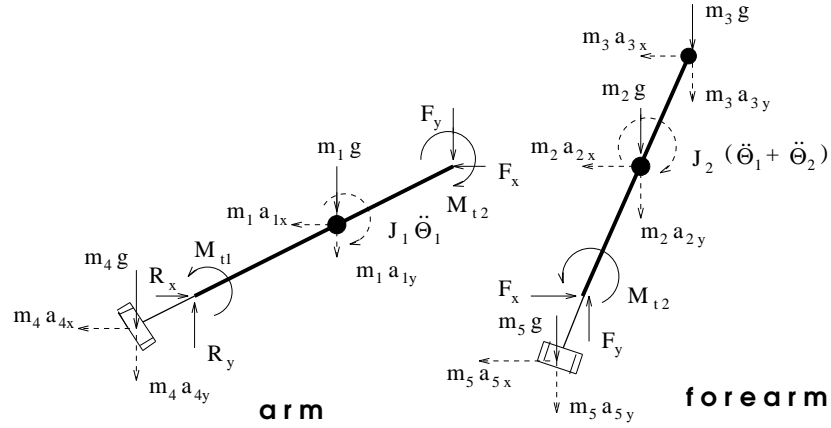


Figure 5.9: Free-body diagrams of the robot arm.

$$D_{22}^1 + D_{21}^2 = 4m_5x_2^2 \sin \theta_0 \cos \theta_0 \quad (5.44)$$

$$D_2 = m_2ge_2 \cos \theta_0 + m_3gL_2 \cos \theta_0 - m_5gx_2 \cos \theta_0 \quad (5.45)$$

where  $J_1$  and  $J_2$  are the rotary inertias of members 1 and 2, respectively. Notation  $\theta_0 = \theta_1 + \theta_2$  is used for convenience.

In addition to the torques, the joint forces are considered in the optimization process. In this application the most convenient way of solving them is to use the force equilibrium conditions in both coordinate directions  $x$  and  $y$ . For this purpose, the free-body diagrams of both members have been depicted in Figure 5.9 (taken from Koski and Osyczka. [196]). The positive directions in this figure are associated with the global  $xy$ -axes, and the positive rotation direction is counterclockwise.

By computing the accelerations from the well-known kinematic equation

$$a_P = a_Q + \alpha \times r_{P/Q} + \omega \times (\omega \times r_{P/Q}) \quad (5.46)$$

analytic expressions for  $a_i(1, \dots, 5)$  can be obtained. Here,  $a_Q$  is the acceleration vector of the comparison point,  $\alpha$  the angular acceleration vector of the member,  $r_{P/Q}$  the position vector from point  $P$  to point  $Q$  along the member, and  $\omega$  the angular velocity of the member. For member 1, point  $Q$  is the support point and  $\alpha = (0, 0, \ddot{\theta}_1)^T$ ,  $\omega = (0, 0, \dot{\theta}_1)^T$ . For member 2, point  $Q$  is at the joint and  $\alpha = (0, 0, \ddot{\theta}_0)^T$ ,  $\omega = (0, 0, \dot{\theta}_0)^T$  where  $\theta_0 = \theta_1 + \theta_2$ . Vector  $r_{P/Q}$  depends on the 5 selected calculation points. Here, the detailed expressions for the accelerations are presented separately, and they also appear as part of the terms in Equation (5.33). The corresponding inertia forces  $m_i a_i$  ( $i = 1, \dots, 5$ ) and the moments  $J_j \alpha_j$  ( $j = 1, 2$ ) with the complete free-body diagrams are shown in Figure 5.9.

The accelerations of the points at which the point masses are located have the following explicit expressions:

$$a_1 = \ddot{\theta}_1 e_1 \begin{bmatrix} -\sin \theta_1 \\ \cos \theta_1 \end{bmatrix} - \dot{\theta}_1^2 e_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} \quad (5.47)$$

$$a_2 = \ddot{\theta}_1 L_1 \begin{bmatrix} -\sin \theta_1 \\ \cos \theta_1 \end{bmatrix} - \dot{\theta}_1^2 L_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} + e_2 \ddot{\theta}_0 \begin{bmatrix} -\sin \theta_0 \\ \cos \theta_0 \end{bmatrix} + \quad (5.48)$$

$$e_2 \dot{\theta}_0^2 \begin{bmatrix} -\cos \theta_0 \\ -\sin \theta_0 \end{bmatrix} \quad (5.49)$$

$$a_3 = \ddot{\theta}_1 L_1 \begin{bmatrix} -\sin \theta_1 \\ \cos \theta_1 \end{bmatrix} - \dot{\theta}_1^2 L_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} + L_2 \ddot{\theta}_0 \begin{bmatrix} -\sin \theta_0 \\ \cos \theta_0 \end{bmatrix} + \quad (5.50)$$

$$L_2 \dot{\theta}_0^2 \begin{bmatrix} -\cos \theta_0 \\ -\sin \theta_0 \end{bmatrix} \quad (5.51)$$

$$a_4 = \ddot{\theta}_1 x_1 \begin{bmatrix} \sin \theta_1 \\ -\cos \theta_1 \end{bmatrix} - \dot{\theta}_1^2 x_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} \quad (5.52)$$

$$a_5 = \ddot{\theta}_1 L_1 \begin{bmatrix} -\sin \theta_1 \\ \cos \theta_1 \end{bmatrix} - \dot{\theta}_1^2 L_1 \begin{bmatrix} \cos \theta_1 \\ \sin \theta_1 \end{bmatrix} + x_2 \ddot{\theta}_0 \begin{bmatrix} \sin \theta_0 \\ -\cos \theta_0 \end{bmatrix} + \quad (5.53)$$

$$x_2 \dot{\theta}_0^2 \begin{bmatrix} \cos \theta_0 \\ -\sin \theta_0 \end{bmatrix} \quad (5.54)$$

Here again the notations  $\theta_0 = \theta_1 + \theta_2$ ,  $\dot{\theta}_0 = \dot{\theta}_1 + \dot{\theta}_2$  and  $\ddot{\theta}_0 = \ddot{\theta}_1 + \ddot{\theta}_2$  have been used. By applying the force equilibrium conditions in the coordinate directions, the following joint reactions (see Figure 5.9) to member 2 are obtained:

$$R_{2x} = m_2 a_{2x} + m_3 a_{3x} + m_5 a_{5x} \quad (5.55)$$

$$R_{2y} = m_2 a_{2y} + m_3 a_{3y} + m_5 a_{5y} + (m_2 + m_3 + m_5)g$$

The torque  $M_{t2}$  at the joint can be calculated from the moment equilibrium condition.

By applying the same routine to member 1, it is possible to derive expressions for the support reactions:

$$\begin{aligned} R_{1x} &= m_1 a_{1x} + m_2 a_{2x} + m_3 a_{3x} + m_4 a_{4x} + m_5 a_{5x} \\ R_{1y} &= m_1 a_{1y} + m_2 a_{2y} + m_3 a_{3y} + m_4 a_{4y} + m_5 a_{5y} + \\ &\quad (m_1 + m_2 + m_3 + m_4 + m_5)g \end{aligned} \quad (5.56)$$

The torque  $M_{t1}$  is obtained again from the moment equilibrium condition. Corresponding to the Lagrangian approach, the torques  $M_{t1}$  and  $M_{t2}$  must be the same as those computed from Equation (5.33).

The resulting support reactions are:

$$R_1 = \sqrt{R_{1x}^2 + R_{1y}^2} \quad , \quad R_2 = \sqrt{R_{2x}^2 + R_{2y}^2} \quad (5.57)$$

The torques  $M_{ti}$  and the forces  $R_i$  are chosen as criteria in the optimization model. It is important to present the detailed expressions for  $M_{ti}$  and  $R_i$  because the choice of the design variables as well as the general complexity of the optimization problem are associated with these formulas.

Given the previous information, now we can formulate the optimization problem. The objective is to find such masses  $m_4$  and  $m_5$  for the counterweights and such joint distances  $x_1$  and  $x_2$  which will minimize the chosen four design criteria. Consequently, the design variable vector is:

$$\bar{x} = (x_1, x_2, x_3, x_4)^T \quad (5.58)$$

where the first two are the distances shown in Figure 5.8,  $x_3 = m_4$  and  $x_4 = m_5$ .

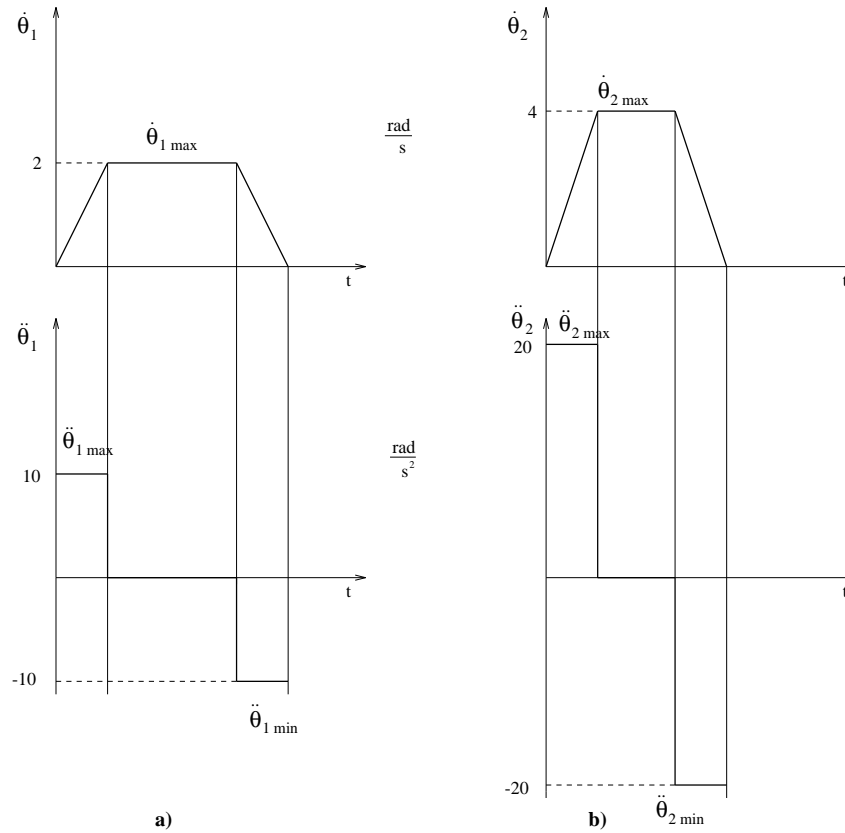


Figure 5.10: Angular velocities and corresponding angular accelerations of the robot arm used for Example No. 8.

The upper and lower limits for all these four design variables can be given in the form

$$x_i^l \leq x_i \leq x_i^u \quad , \quad i = 1, \dots, 4 \quad (5.59)$$

The torques  $m_{t1}$  and  $M_{t2}$  at the arm joints are chosen as the first two criteria of the vector objective function. Their minimization is important because it is then possible to use smaller motors, and the energy consumption is lower if the variation ranges of the torques are small [196]. In the explicit expressions of Equation (5.33), terms  $m_4 x_1$ ,  $m_4 x_1^2$ ,  $m_5 x_2$ , and  $m_5 x_2^2$  appear, and thus it is reasonable to choose the design variables in the way presented.

The torques do not depend on the design variables alone, but also on the position of the robot arm  $(\theta_1, \theta_2)$ . On the angular velocities  $(\dot{\theta}_1, \dot{\theta}_2)$  and on the angular accelerations  $(\ddot{\theta}_1, \ddot{\theta}_2)$ . Usually, the working space of the robot arm is restricted, and thus constraints are needed of the following form:

$$\theta_i^l \leq \theta_i \leq \theta_i^u \quad , \quad i = 1, 2 \quad (5.60)$$

Here,  $\theta_i^l$  and  $\theta_i^u$  are the lower and the upper limits of the angles  $\theta_i$ . In each position of the arm, the angular velocities and accelerations may be different. In order to optimize the performance of the robot, the torques should be as small as possible at all working positions and at all existing angular velocity acceleration combinations. Thus, the first two criteria are chosen as follows:

$$\begin{aligned} f_1(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\theta_i, \dot{\theta}_i} M_{t1} \\ f_2(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\theta_i, \ddot{\theta}_i} M_{t2} \end{aligned} \quad (5.61)$$

where notation  $\dot{\theta}_i, \ddot{\theta}_i$  is associated with the chosen angular velocity profile. This is shown in Figure 5.10 (taken from Koski and Osyczka. [196]) where a trapezoidal profile, typical of robot applications, has been depicted for both members. The corresponding angular accelerations are also presented in this figure.

The construction of joints, especially with the choice of bearings, depends largely on the reaction forces at the joints. Thus, it seems reasonable to choose the maximum values of the joint forces as two additional criteria.

By using the fixed trapezoidal velocity profiles (see Figure 5.10) and every feasible position of the arm, these criteria can be expressed in the form

$$\begin{aligned} f_3(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\theta_i, \theta_i} R_1 \\ f_4(\bar{x}) &= \max_{\theta_1} \max_{\theta_2} \max_{\theta_i, \theta_i} R_2 \end{aligned} \quad (5.62)$$

The geometrical interpretation of all the four criteria is the following [196]: a small movement at every position  $(\theta_1, \theta_2)$  of the arms is performed by using the fixed trapezoidal profiles  $(\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1, \ddot{\theta}_2)$  shown in Figure 5.10), and the maximum values of the torques and the joint forces during the movement are selected.

By using the design variables  $x_i$  given in Equation (5.29), the criteria presented in Equations (5.61) and (5.62), the side constraints of (5.59), and the state constraints of (5.60), it is now possible to formulate the multicriteria optimization problem [196]:

$$\min (f_1(\bar{x}), f_2(\bar{x}), f_3(\bar{x}), f_4(\bar{x}))^T \quad (5.63)$$

subject to

$$\begin{aligned} \theta_i^l &\leq \theta_i \leq \theta_i^u & i = 1, 2 \\ x_i^l &\leq x_i \leq x_i^u & i = 1, 2, 3, 4 \end{aligned} \quad (5.64)$$

The numerical design data for the design problem is given below [196]. These values are close to the arms of the PUMA-560 robot shown in Figure 5.7 [195].

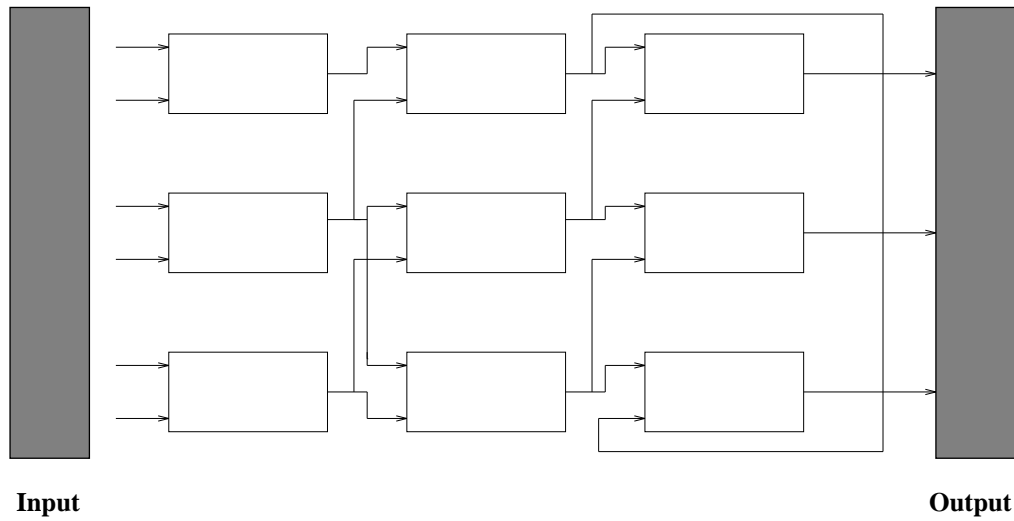


Figure 5.11: A gate in a two-dimensional template, gets its second input from either one of two gates in the previous column.

$$\begin{aligned}
 m_1 &= 17 \text{ kg}, & m_2 &= 6 \text{ kg}, & m_3 &= 2 \text{ kg}, \\
 L_1 &= L_2 = 0.43 \text{ m}, & e_1 &= 0.07 \text{ m}, & e_2 &= 0.14 \text{ m}, \\
 \theta_1^l &= -40^\circ, & \theta_1^u &= 220^\circ, & \theta_2^l &= -140^\circ, & \theta_2^u &= 140^\circ, \\
 \dot{\theta}_{1 \text{ max}} &= 2 \frac{\text{rad}}{\text{s}}, & \dot{\theta}_{2 \text{ max}} &= 4 \frac{\text{rad}}{\text{s}}, \\
 \ddot{\theta}_{1 \text{ max}} &= 10 \frac{\text{rad}}{\text{s}^2}, & \ddot{\theta}_{2 \text{ max}} &= 20 \frac{\text{rad}}{\text{s}^2}, \\
 x_1^l &= x_2^l = 0, & x_1^u &= x_2^u = 0.2 \text{ m}, & x_3^l &= x_4^l = 0, \\
 x_3^u &= 35 \text{ kg}, & x_4^u &= 15 \text{ kg}, & J_1 &= 0.2619 \text{ kg} \cdot \text{m}^2, & J_2 &= 0.0924 \text{ kg} \cdot \text{m}^2
 \end{aligned}
 \tag{5.65}$$

## 5.8 Example 8 : Design of a combinatorial circuit

Consider the instantiation of the combinatorial circuit design problem known as the **adder problem** [197]. We want to find the combination of five



possible types of gates (AND, OR, NOT, XOR and WIRE) so that our circuit performs the two-bit addition of its inputs. The objective is to minimize the use of gates other than WIRE, while keeping a functional design. The circuit can be seen as a two-dimensional array of gates  $S$ , where a gate  $S_{i,j}$  gets its first input from  $S_{i,j-1}$  and its second input from either  $S_{i+1,j-1}$  or  $S_{i-1,j-1}$  as shown in Figure 5.11 (taken from Louis and Rawlins [197]).

## 5.9 Summary and additional comments

In this last section I will summarize the characteristics of the eight problems introduced in this chapter, adding some of the reasons why they were chosen to test MOSES.

1. Design of an I-beam: This problem has 4 design variables, 1 non-linear constraint and 2 objective functions (both to be minimized). Due to the simplicity of the analysis that this problem requires, it is a good choice to tune up MOSES. Also, since it has only 2 objectives and its variables have relatively small ranges, it is possible to graph both the feasible region and the Pareto front in a reasonable amount of time.
2. Machining recommendations: This problem has 3 design variables, 3 linear constraints and 4 objective functions (all to be maximized). No graph can be produced of the feasible region or the Pareto front because of the high number of objectives. The main importance of this problem is that the ideal vector is achievable.
3. Design of a machine tool spindle: This problem has 4 design variables, 9 linear and 1 non-linear constraint and 2 objective functions (both to be minimized). The feasible region and the Pareto front can be graphically

displayed in a reasonable amount of time. One of the interesting issues when dealing with this problem is that 2 of its design variables are discrete, which makes it hard (or even impossible to deal with) for certain mathematical programming techniques.

4. Design of a 10-bar plane truss: This problem has 20 design variables, 22 non-linear constraints and 3 objective functions (all to be minimized). The analysis involved in this problem is quite complex and takes a considerable amount of CPU time. Also, since the range of each variable is quite large, the feasible region can not be displayed graphically in a reasonable amount of time. Such a large search space, together with the fact that the objective functions can not be provided explicitly and that they are highly conflicting (making it difficult to produce good trade-offs), make this an interesting problem.
5. Design of a 25-bar space truss: This problem has 8 design variables, 55 non-linear constraints and 3 objective functions (all to be minimized). The objective functions can not be provided explicitly, and they are highly conflicting among each other. The range of each design variable is quite large, and therefore the feasible region can not be displayed in a reasonable amount of time. This problem has some similarities with the previous one, although its analysis is different. One of its particular remarks is that it has less design variables, but a higher constrained search space. This should favor mathematical programming techniques.
6. Design of a 200-bar plane truss: This problem has 29 design variables, 200 non-linear constraints and 3 objective functions (all to be minimized). The

objective functions can not be provided explicitly, and they are highly conflicting among each other. The range of each design variable is quite large, and therefore the feasible region can not be displayed in a reasonable amount of time. This problem has the highest number of design variables of all, which will make it extremely difficult for mathematical programming techniques and the GA using binary representation. Due to its size, much CPU time has to be spent in the analysis stage of this problem.

7. Design of a robot arm: This problem has 4 design variables, no constraints and 4 objective functions (all to be minimized). The analysis stage of this problem requires extremely high amounts of CPU time. The values of the objectives are highly sensitive to the values of the design variables, which makes this a difficult problem for any GA-based technique. Also, the fact that we are dealing with an unconstrained search space, mathematical programming techniques will have some difficulties to find a good trade-off.
8. Design of a combinatorial circuit: This problem has 16 design variables, 48 constraints and 2 objective functions (both to be maximized). This problem involves a broader concept of design than the one encompassed by previous examples. Here, the emphasis is on generating functional circuits minimized with respect to the use of certain gates. Consequently, less numerical manipulations are involved, but a more difficult search space is faced. Because of the way in which this problem is stated, only GA-based techniques can be used to solve it.

## Chapter 6

# Analysis of Results

In this chapter, I will present the results obtained using each one of the methods implemented in MOSES (see Chapter 4) to solve the problems presented in Chapter 5. I will start by analyzing the complexity of each one of the multi-objective optimization algorithms employed. Also, interesting issues such as the closeness of the best solution generated to the ideal vector and the final distribution of Pareto points will be analyzed. Additionally, I will present some details on the computation of the ideal vector using both mathematical programming techniques and the genetic algorithm. In this respect, there are some surprising results, since the GA was able to find better results than the mathematical programming techniques used, applying the procedure described in Chapter 4 to adjust its parameters, and using floating point representation. These results are consistent and appear in every single problem. These results suggest that the GA may be a practical numerical optimization tool. Finally, I will critique each of these algorithms, describing their advantages and disadvantages based on the experiments run on the examples presented in Chapter 5.

## 6.1 Measuring the Complexity of each Algorithm

I will start by analyzing the complexity of the Simple Genetic Algorithm (SGA) presented by Goldberg in his book [145]. The basic outline of the algorithm is the following:

```

During g generations do:
  for m do:
    decode();
    compute_fitness();
    select();
    crossover();
    mutation();
  end loop for m
  statistics();
end loop for g

```

Here, **g** refers to the number of generations and **m** refers to the population size.

The time taken for decoding depends on the representation scheme used. When binary representation is used, it requires  $2 \times l + p$  operations, where  $l$  is the length of the string and  $p$  is the number of variables. The reason is that transforming a binary string into a decimal number requires  $l$  raised to the second power plus  $l$  additions. Furthermore, we need to perform one division per each variable in order to adjust the number to the precision desired. So, as the length of the chromosome increases, the decoding takes more time. When a floating

point representation is used, we need  $l - 1$  additions and  $l$  divisions. Since the strings are normally shorter under this representation scheme, and it is faster to divide than to raise to the second power, this encoding is more efficient, mainly when there are many design variables involved.

To compute a single fitness value is really the main issue when measuring time complexity of a GA, because many times in multiobjective optimization the functions involved are very complex and a considerable amount of time could be needed just to evaluate once the fitness of a chromosome, as we will see in some of the examples under study in this work.

The time complexity of selection has been studied by Goldberg and Deb [150]. If we use **roulette wheel selection**, we simulate the spin of a weighted roulette wheel. Therefore, if the search for such location is performed via linear from the beginning of the list, each selection requires  $O(m)$  steps, where  $m$  refers to the population size, because on average half the list will be searched. Now, since in each generation  $m$  spins are required to fill the population, this selection strategy requires  $O(m^2)$  steps. Using binary search, we can improve the efficiency of the algorithm to  $O(m \log m)$ , if we can afford the extra memory required, because binary search is  $O(\log m)$ .

If we use **stochastic remainder selection**, the expected number of copies of a string is calculated as  $c_i = \frac{f_i}{f}$ , and the integer portions of the count are assigned deterministically. The remainders are then used probabilistically to fill the population. If we fill without replacement, each remainder is used to bias the flip of a coin that determines whether the structure receives another copy or not, and then the algorithm is  $O(m)$ , because only one pass is required. If we use replacement, the remainders are used to size the slots of a roulette wheel selection process, so the algorithm takes in this case  $O(m^2)$  steps.

The so-called **stochastic remainder selection** is performed by sizing the slots of a weighted roulette wheel equally spaced markers along the outside of the wheel, and spinning the wheel once; the number of copies an individual receives is then calculated by counting the number of markers that fall in its slot. This algorithm is  $O(m)$  because only one pass is needed through the list after the sum of the function values is calculated.

**Ranking selection** is a two-step process in which the chromosomes are first sorted and then their assignment values are used in some form of proportionate selection. Since sorting can be done in  $O(m \log m)$  using quicksort, and proportionate selection varies from  $O(m)$  to  $O(m^2)$ , this technique has a complexity between  $O(m \log m)$  and  $O(m^2)$  depending upon the proportionate selection scheme used.

**Tournament selection** requires the random selection of a constant number of individuals from the population. Since the comparison among those individuals can be done in constant time and  $m$  competitions are required to fill a generation, the algorithm is  $O(m)$ .

Crossover and mutation only require  $l$  manipulations of the string, but since those are done in memory, there is no significant time to count for.

Finally, to get the population statistics, we need  $m$  comparisons, since we have to determine the best overall individual.

All the previous analysis shows that the most significant operations during the execution of a GA are selection, decoding and fitness evaluation. Since the GA runs a nested loop, the basic process requires  $g \times m$  fitness evaluations assuming everything inside the loop is done in constant time. If we use a selection strategy that takes linear time, then the GA will take  $g \times m^2$  plus the time needed for decoding. Considering a conservative population policy in which the number of

individuals is at least twice the length of the string, the time needed for decoding is about  $O(n^2)$ , too. Thus, in general, we may say that the GA is  $O(m^2)$ . Observe that  $g$  may vary a lot, so in certain cases the algorithm could become  $O(m^3)$ . Nevertheless, if  $g$  is greater than  $m$ , the value of  $m^2$  will, in general, be much larger than  $g$ , so the performance of the algorithm will still be dominated by the  $O(m^2)$  complexity. This, of course, could get slower if the fitness function requires too much time. For example, if a matrix inversion is required at each step of the GA, we have to consider that an  $O(p^3)$  algorithm (where  $p$  is the size of the matrix) is going to be executed inside a nested loop, and that is going to take longer than the search itself. In all the following GA-based methods, binary tournament selection was used unless otherwise is specified.

### 6.1.1 Monte Carlo Methods

In Chapter 4 I described two exploratory techniques that use Monte Carlo methods to find the min-max optimum solution. The first explores the variable space twice. Therefore we can analyze its complexity by decomposing the algorithm in two parts:

1. Generate ideal vector: At this stage,  $m$  points are randomly generated (where  $m$  is defined by the user and is equivalent to the population size used with the GA). For each solution, we have to check feasibility and evaluate each one of the objective functions. This means that at least  $m \times k$  evaluations are needed (where  $k$  is the number of objective functions of the problem).
2. Compute the min-max optimum: We have to generate again  $m$  random solutions and check their feasibility. This also requires  $m \times k$  function evaluations in the best case (if a solution is not feasible we have to generate it and check it again). Then, we have to compute the deviations of each one of these



solutions with respect to the ideal vector. This requires  $m \times k$  operations because we have to do that for each objective. Finally, the best solution has to be found (the solution with the minimum deviation from the ideal vector). This last process requires  $m$  comparisons.

From the previous analysis, we can see that we need to perform at least  $3 \times m \times k + m$  operations. Since  $k$  is normally very small compared to  $m$ , then the complexity of this algorithm can be considered linear.

The second Monte Carlo method used by MOSES is faster, since it only requires looking into the solution space once. In this case we still require  $m \times k$  operations to generate the ideal vector, but after that, we check if the solution generated is in the Pareto set. This requires another  $m$  comparisons. Finally, we apply the min-max algorithm over the Pareto optimum solution found to determine the min-max optimum. As indicated before, we require  $r \times k$  operations to compute the deviations of each solution with respect to the ideal vector, and then  $r$  comparisons to find the min-max optimum ( $r$  is the number of Pareto optimum solutions). Since  $r$  will always be less than  $m$ , we require less time for this process than when the previous method is used. This second method is also linear, but requiring about half of the operations of the previous method. However, as indicated in Chapter 4, its main drawback is that it requires more memory than the previous method, since it has to store the entire Pareto set.

### 6.1.2 Osyczka's Multiobjective Optimization System

This is a fairly complicated piece of software and it is not easy to analyze it. Basically, it consists of two stages. The first stage encompasses the computation of the ideal vector using single optimization techniques applied to each vector (the user can also provide the target values). In my experiments I only used the flexible

tolerance method to compute the ideal vector, and therefore I will provide more details of how it works to have a better idea of how to measure its complexity. The complete details and proofs of its convergence may be found in Himmelblau [185].

The basic idea of the Flexible Tolerance method is to improve the value of the objective function by using information provided by feasible points, as well as certain nonfeasible points termed **near-feasible points**. The near-feasibility limits are gradually made more restrictive as the search proceeds toward the solution of the problem, until in the limit only feasible solutions are accepted. My implementation uses the flexible polyhedron search of Nelder and Mead [185], but any other unconstrained minimization algorithm could replace it. Let us assume that  $\bar{x}^{(0)}$  is our initial guess, that  $t = 0.05\Phi^{(k)}$  is the size of the initial polyhedron,  $m$  is the number of equality constraints,  $n$  is the number of variables,  $r = n - m$  is the number of degrees of freedom,  $k = 0, 1, \dots$  is an index referring to the number of completed stages of search,  $\Phi$  is the tolerance criterion,  $\alpha = 1$ ,  $\beta = 0.5$ ,  $\gamma = 2$ ,  $\epsilon = 1 \times 10^{-5}$ ,  $s$  is the current stage and:

$$T(\bar{x}) = \left[ \sum_{i=1}^m h_i^2(\bar{x}) + \sum_{i=m+1}^p U_i g_i^2(\bar{x}) \right]^{\frac{1}{2}} \quad (6.1)$$

where  $U_i$  is the Heaviside operator [185] such that  $U_i = 1$  for  $g_i(\bar{x}) \geq 0$  and  $U_i = 0$  for  $g_i(\bar{x}) < 0$ . Also,  $h_i(\bar{x}) = 0$  for  $i = 1, \dots, m$  and  $g_i(\bar{x}) \geq 0$  for  $i = m + 1, \dots, p$  are respectively the equality and inequality constraints of the problem.

Additionally,

$$\Phi^{(0)} = 2(m+1)t \quad (6.2)$$

and

$$\Phi^{(k)} = \min \left\{ \Phi^{(k-1)}, \frac{m+1}{r+1} \sum_{i=1}^{r+1} \left\| \bar{x}_i^{(k)} - \bar{x}_{r+2}^{(k)} \right\| \right\} \quad (6.3)$$

The procedure for obtaining the vertices  $\bar{x}_i^{(0)}$ , for  $i = 1, \dots, r+1$ , required to start the search is as follows. Compute  $\Phi^{(0)}$  and compute the value of  $T(\bar{x})$  at the initial vector  $\bar{x}^{(0)}$ . As could be seen before,  $T(\bar{x})$  is defined as the positive square root of the sum of the squared values of all the violated equality and/or inequality constraints of the problem. If  $T(\bar{x}^{(0)}) \leq 0$ , then  $\bar{x}^{(0)}$  is a feasible or near-feasible point and the initial vertices,  $\bar{x}_i^{(0)}$ , for  $i = 1, \dots, r+1$ , are obtained using the unconstrained method of Nelder and Mead [185]. If  $T(\bar{x}^{(0)}) > \Phi^{(0)}$ ,  $T(\bar{x})$  is minimized until a feasible or near-feasible  $\bar{x}$  vector becomes the base point for building the initial polyhedron. The complete algorithm is the following:

1. Is  $T(\bar{x}_i^{(k)}) \leq \Phi^{(k)}$ ? If yes, then go to step 2, otherwise go to step 3.
2. Determine  $\bar{x}_{r+2}^{(k)} = \frac{1}{r} \left[ \left( \sum_{i=1}^{r+1} \bar{x}_i^{(k)} \right) - \bar{x}_h^{(k)} \right]$ . Go to step 4.
3. Minimize  $T(\bar{x}^{(k)})$  so that  $T(\bar{x}_i^s) \leq \Phi^{(k)}$ . Let  $\bar{x}_i^{(k)} = \bar{x}_i^s$ . Compute  $f(\bar{x}_i^{(k)})$ .  
Go to step 1.
4. Is  $\Phi^{(k)} \leq \epsilon$ ? If yes, then STOP, otherwise determine  $\bar{x}_{r+3}^{(k)} = \bar{x}_{r+2}^{(k)} + \alpha(\bar{x}_{r+2}^{(k)} - \bar{x}_h^{(k)})$ .
5. Is  $T(\bar{x}_{r+3}^{(k)}) \leq \Phi^{(k)}$ ? If yes, then go to step 7, otherwise go to step 6.
6. Minimize  $T(\bar{x}_{r+3}^{(k)})$  so that  $T(\bar{x}_i^s) \leq \Phi^{(k)}$ . Let  $\bar{x}_{r+3}^{(k)} = \bar{x}_i^s$ . Compute  $f(\bar{x}_{r+3}^{(k)})$ .

7. Is  $f(\bar{x}_{r+3}^{(k)}) < f(\bar{x}_h^{(k)})$ ? If yes, then go to step 19, otherwise go to step 8.
8. Is  $f(\bar{x}_{r+3}^{(k)}) < f(\bar{x}_s^{(s)})$ ? If yes, then go to step 9, otherwise go to step 10.
9. Determine  $\bar{x}_h = \bar{x}_{r+3}^{(k)}$ ,  $f(\bar{x}_h^{(k)}) = f(\bar{x}_{r+3}^{(k)})$ . Make  $k = k + 1$  and go to step 1.
10. Is  $f(\bar{x}_{r+3}^{(k)}) < f(\bar{x}_h^{(k)})$ ? If yes, then go to step 11, otherwise go to step 12.
11. Determine  $\bar{x}_h^{(k)} = \bar{x}_{r+3}^{(k)}$ .
12. Determine  $\bar{x}_{r+5}^{(k)} = \bar{x}_{r+2}^{(k)} + \beta(\bar{x}_h^{(k)} - \bar{x}_{r+2}^{(k)})$ .
13. Is  $T(\bar{x}_{r+5}^{(k)}) \leq \Phi^{(k)}$ ? If yes, then go to step 15, otherwise go to step 14.
14. Minimize  $T(\bar{x}_{r+5}^{(k)})$  so that  $T(\bar{x}_i^{(s)}) \leq \Phi^{(k)}$ . Let  $\bar{x}_{r+5}^{(k)} = \bar{x}_i^{(s)}$ . Compute  $f(\bar{x}_{r+5}^{(k)})$ .
15. Is  $f(\bar{x}_{r+5}^{(k)}) < f(\bar{x}_h^{(k)})$ ? If yes, then go to step 16, otherwise go to step 17.
16. Determine  $\bar{x}_h^{(k)} = \bar{x}_{r+5}^{(k)}$ ,  $f(\bar{x}_h^{(k)}) = f(\bar{x}_{r+5}^{(k)})$ , and  $k = k + 1$ . Go to step 1.
17. Determine  $\bar{x}_i^{(k)} = \bar{x}^{(k)} + 0.5(\bar{x}_i^{(k)} - \bar{x}_l^{(k)})$ , for  $i = 1, \dots, r + 1$ .
18. Determine new values for  $f(\bar{x}_i^{(k)})$ , for  $i = 1, \dots, r + 1$ , and  $k = k + 1$ . Go to step 1.
19. Determine  $\bar{x}_{r+4}^{(k)} = \bar{x}_{r+3}^{(k)} + \gamma(\bar{x}_{r+3}^{(k)} - \bar{x}_{r+2}^{(k)})$ .
20. Is  $T(\bar{x}_{r+4}^{(k)}) \leq \Phi^{(k)}$ ? If yes, then go to step 22, otherwise go to step 21.
21. Minimize  $T(\bar{x}_{r+4}^{(k)})$  so that  $T(\bar{x}_i^{(s)}) \leq \Phi^{(k)}$ . Let  $\bar{x}_{r+4}^{(k)} = \bar{x}_i^{(s)}$ . Compute  $f(\bar{x}_{r+4}^{(k)})$ .
22. Is  $f(\bar{x}_{r+4}^{(k)}) \leq f(\bar{x}_l^{(k)})$ ? If yes, then go to step 24, otherwise go to step 23.
23. Determine  $\bar{x}_h^{(k)} = \bar{x}_{r+3}^{(k)}$ ,  $f(\bar{x}_h^{(k)}) = f(\bar{x}_{r+3}^{(k)})$ , and  $k = k + 1$ . Go to step 1.
24. Determine  $\bar{x}_h^{(k)} = \bar{x}_{r+4}^{(k)}$ ,  $f(\bar{x}_h^{(k)}) = f(\bar{x}_{r+4}^{(k)})$ , and  $k = k + 1$ . Go to step 1.

As it has been reported by Himmelblau [185], the previous algorithm presents a great variation in terms of performance, and it is very sensitive to the guess provided by the user and to the values of  $\alpha$ ,  $\beta$  and  $\gamma$ . A good discussion of such issues is shown in Himmelblau's book [185], and the algorithm could require until 50 % more time to achieve convergence if such parameters are not properly set. From my own experience, I can tell that keeping these last 3 parameters fixed (as suggested by Himmelblau) and using a good guessing point, the technique is very fast, but it still requires about 200 % or 300 % more time than any Monte Carlo method. If a bad guessing point is provided this time could be increased until 100 % more (i.e., 400 % slower than any Monte Carlo method).

The second part of Osyczka's system consists of finding the min-max optimum. As I explained before, this can be done in linear time. Therefore, the search of the ideal vector turns out to be the most time consuming part of the program. This extra time complexity is justified by the fact that we are using an adaptive search technique that will be able to move the search polyhedron to the feasible zone even when an infeasible starting point is provided, and it is expected to provide better results than a random search technique. Nevertheless, we will see from the results obtained with the examples included in Chapter 5 that Osyczka's system is a very useful tool only when we can provide very good guesses. In practice, it is therefore advisable to use a random search technique to get an initial solution and then to apply any of the methods of Osyczka's system to fine tune the results.

### 6.1.3 Lexicographic Method and Linear Combination

The Lexicographic method and a simple linear combination of objectives both use the GA in its simplest form, so their complexity remains the same as

indicated before, depending on the selection strategy used. In my experiments I always used binary tournament selection for both methods, which is the most efficient selection strategy available. Therefore, these two techniques are quite fast and are the lower bound in terms of time of all the GA-based techniques.

#### 6.1.4 VEGA

Schaffer's VEGA requires the same complexity than the SGA, because it only subdivides the population into  $k$  subgroups, where  $k$  is the number of objectives, and applies the corresponding objective function to each one of them. In my implementation, I run a loop from 1 to  $k$ , and inside that loop I generate  $p$  individuals, where  $p = \text{popsize}/k$ . This is equivalent to running a single loop from 1 to  $\text{popsize}$ . The only additional modification in the code is the addition of a `switch` statement to decide what objective function to use at each fitness function call. This requires a flag to indicate which objective has to be evaluated at each time, but does not increase the time complexity of the algorithm in a significant way (only an extra comparison is required at each fitness function invocation).

#### 6.1.5 NSGA

This is the slowest algorithm included in MOSES, mainly because of the way in which it was implemented. The SGA had to be modified in the procedure `generation()` as to include the algorithm described by Srinivas [181]. Therefore, we have to add to the time complexity of the SGA, the time required by this additional code. Such extra code consists of four parts that can be analyzed separately:

1. Compute vector values for each chromosome: Since we have to manipulate the objective function values of each chromosome, for the sake of efficiency, I

decided to sacrifice some memory, and store such values at the beginning of the algorithm. This requires to compute them only once, instead of having to do it each time that it is required. This process requires to evaluate  $k$  objective function values  $m$  times (where  $m$  is the population size).

2. Check dominance: This requires three nested loops. The first consists of the number of fronts (same as  $k$  in my implementation); the second and third consist of the population size. Therefore, we need to perform  $k \times m^2$  operations.
3. Compute fitness distances: To do sharing, we need to know first how far are the solutions one from another. This requires a doubly nested loop from 1 to  $m$ , since we need to compute our distance metrics on each individual against all the others in the population. Thus, this process requires  $m^2$  operations.
4. Do sharing: The last stage of the process consists of doing sharing by counting how many individuals are in each nest, and decreasing the fitness of each individual on that proportion. This process also requires  $m^2$  operations.

From the previous analysis, we can see that the second process (checking for dominance) is the one that requires more operations. However, that is not really the slowest process within this algorithm, because normally it is more time consuming to evaluate the objective functions. Nevertheless, since such function evaluation is already included in the analysis of the SGA, we should count only for the extra time required by steps 2–4. This implies that this algorithm is  $O(c \times m^2)$ , where  $c = k + 2$ . The value of  $k$  is normally very small compared to the value of  $m$ , so the time complexity is really dominated by the  $m^2$  operations. Finally, I should also mention that this method uses stochastic remainder selection, which

also increases the time complexity of the algorithm. This contributes to make it the slowest GA-based algorithm used in my experiments.

### 6.1.6 MOGA

This implementation is faster than Srinivas' NSGA, but also has some time consuming processes. First, the algorithm requires computing the vector values (i.e., the function values for all objectives) for each chromosome. This requires  $k \times m$  function evaluations. Then, we need to check for dominance. The loop in this case is different from what Srinivas uses, but the time complexity remains the same, since we still require  $k \times m^2$  operations to assign ranks. The next stage of the algorithm consists of computing the area for  $\sigma_{share}$ . This requires  $k^2 + k \times m$  operations. Finally, we have to compute distances and do sharing, as in the previous algorithm. We need  $m^2$  operations for each of these two processes.

Fonseca's MOGA requires  $k \times m^2 + k^2 + 2m^2 + k \times m$  operations. This algorithm can be considered  $O(c \times m^2 + p)$ , where  $c = k + 2$  and  $p = k \times m + k^2$ . This seems to be a slower algorithm than NSGA, but that is not necessarily true, because the use of binary tournament selection in this case requires a linear algorithm for that stage of the algorithm, instead of a quadratic algorithm as in NSGA. That accounts for the extra (almost linear) time complexity required to compute  $\sigma_{share}$ , which by the way, eliminates the need of extra experimentation to try to find its optimum value. That is the reason why this method turns out to be faster than NSGA in practice.



### 6.1.7 NPGA

The implementation of this algorithm also requires computing the vector values for each chromosome, which implies  $k \times m$  function evaluations. Additionally, this implementation requires a modification of the function `select()` so that a local dominance procedure is run. The main loop required to check for local dominance goes from 2 to  $t_{dom} + 2$ . Since the value of  $t_{dom}$  is normally small (assumed 10 in my experiments), then we need to execute this loop for only a few times. The worst behavior of the algorithm is exhibited when we have to do sharing, in which case we need to execute  $2 \times k \times m + 2 \times m$  operations. All these operations have to be done for each individual in the population; thus, the complexity of the algorithm is  $O(c \times m + p)$  in the worst case, where  $c = t_{dom}$  and  $p = (k + 2) \times m^2$ . When sharing is not necessary (i.e., when one individual totally dominates or is totally dominated) only  $c \times m$  operations are required. This analysis shows why this method is much faster than the two previous techniques described. However, some additional issues such as finding the proper value of  $t_{dom}$  and  $\sigma_{share}$  may make it necessary to run this algorithm more than once. It should be nevertheless mentioned that in my experiments, I found the behavior of this algorithm to be very robust when  $t_{dom} = 10$  and  $\sigma_{share} = 0.1$  were used, but we will also see how such robustness is seriously affected when these parameters are changed.

### 6.1.8 Hajela's Method

This method is very efficient, since the only modification that the SGA implementation requires is the addition of a process to compute the metric distances for the niches, and another to do sharing. Each one of these processes requires  $O(m^2)$  operations, as indicated before, but no further processing is required,

since only an additional decision is required to implement mating restrictions. This method is, therefore, very efficient and easy to implement, although it has the drawback of requiring longer strings to place the weights (a factor that increases the decoding time within the GA itself) and that we require to know the ideal vector. This last requirement may be satisfied if we have certain knowledge of the problem and it is possible to estimate the goals that we want to achieve. An exploratory method such as the Monte Carlo strategies described in Chapter 4 can be used for such purpose.

### 6.1.9 My Weighted Min-Max Method

It is very easy to compute the complexity of this method, since all it does is to spawn as many processes as weights are provided by the user. Each one of these processes is just a simple GA like the one described above, so we only have to multiply its processing time by  $W$  (where  $W$  is, of course, the number of weights given by the user). As the examples that follow show, the number of weights does not have to be excessive. In all the following examples, no more than 20 weights were used to produce excellent results. Moreover, since each process is completely independent from the other (they only share a file that contains the pareto solutions), then the method can be easily implemented in parallel or in a distributed system to improve its performance, becoming therefore as fast as a conventional SGA. The only change in the code is done in the fitness function, that requires the computation of the deviations of each objective with respect to the ideal vector. It is important to point out that if such an ideal vector is not known beforehand, we can provide any reasonable target values that we want to achieve, and the technique will still work very well. If the proper resources are available (i.e., several workstations in a distributed system, or a parallel architecture), this

method requires the shortest amount of time, together with the Lexicographic method and the linear combination of objectives. However, if such facilities are not at hand, it could become one of the slowest, because we would need to execute sequentially the same program several times (normally between 10 and 20).

### 6.1.10 My Min-Max Strategy with Sharing

My implementation modifies the function `select()` and the procedure `statistics()`. The changes in `select()` are somewhat similar to those performed within NPGA, but now instead of using a fixed  $t_{dom}$  to determine local dominance, I use a min-max strategy to check for local dominance over the entire population. My algorithm requires to loop from 1 to  $k$  to compute the deviations of the min-max method for each chromosome, and when there are ties, sharing is performed over the same basis as in NPGA. Then, we require  $k \times m + 2 \times (k+2) \times m^2$  operations when sharing is necessary. Otherwise, the algorithm will be  $O(k \times m)$ . This is significantly faster than most of the other GA-based algorithms, but it also requires a way of finding  $\sigma_{share}$ , which is not necessarily the same as before. Furthermore, we also need to compute the local ideal vector at each generation, with a process that requires  $k \times m$  operations (for small  $k$  and large  $m$  it can be considered linear). But even after performing these additional computations, this method is one of the fastest GA-based strategies with which I experimented, although its main weakness is its high sensitivity to the value of  $\sigma_{share}$ .

## 6.2 Example 1 : Design of an I-Beam

Before I start the analysis of results for this example, it is convenient to show the feasible region in the objective function space, so that we can visualize the Pareto front that we wish to achieve. Figure 6.3 shows the feasible region

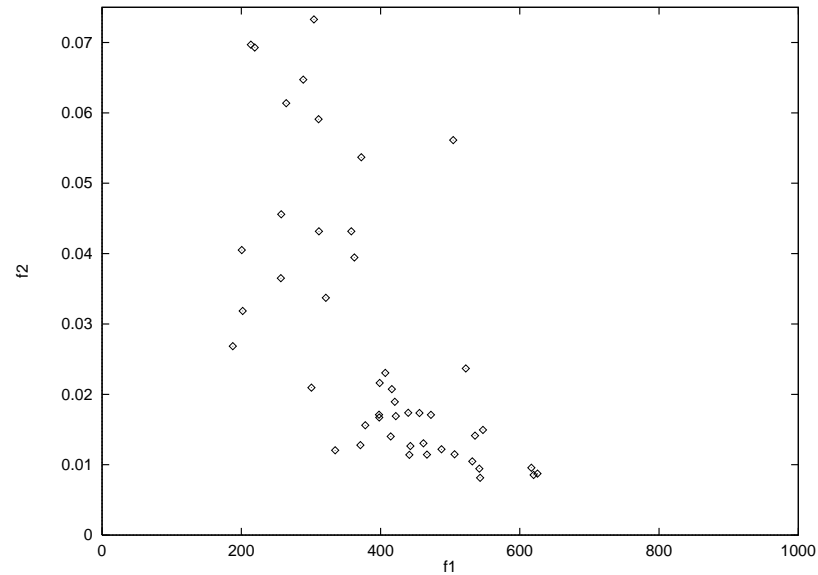


Figure 6.1: Example 1: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed.

Method	$x_1$	$x_2$	$x_3$	$x_4$	$f_1$	$f_2$
Monte Carlo 1	30.84	28.26	3.79	4.06	<b>188.65</b>	0.06175
Monte Carlo 1	52.97	44.08	1.99	0.99	555.22	<b>0.00849</b>
Min-Max (OS)	74.97	44.97	1.97	1.97	<b>316.85</b>	0.01697
Min-Max (OS)	74.99	44.99	1.99	2.06	326.49	<b>0.01636</b>
GA (Binary)	66.39	38.63	0.90	0.91	<b>128.27</b>	0.05241
GA (Binary)	80.00	50.00	4.99	4.99	848.41	<b>0.00591</b>
GA (FP)	61.14	41.14	0.90	0.90	<b>127.46</b>	0.06034
GA (FP)	80.00	50.00	5.00	5.00	850.00	<b>0.00590</b>
Literature	60.70	49.90	0.90	0.90	<b>128.47</b>	0.060
Literature	80.00	50.00	5.00	5.00	850.00	<b>0.0059</b>

Table 6.1: Comparison of results computing the ideal vector of example 1 from Chapter 5 (design of an I-beam). For each method the best results for optimum  $f_1$  and  $f_2$  are shown in **boldface**. OS stands for Osyczka's Multiobjective Optimization System.

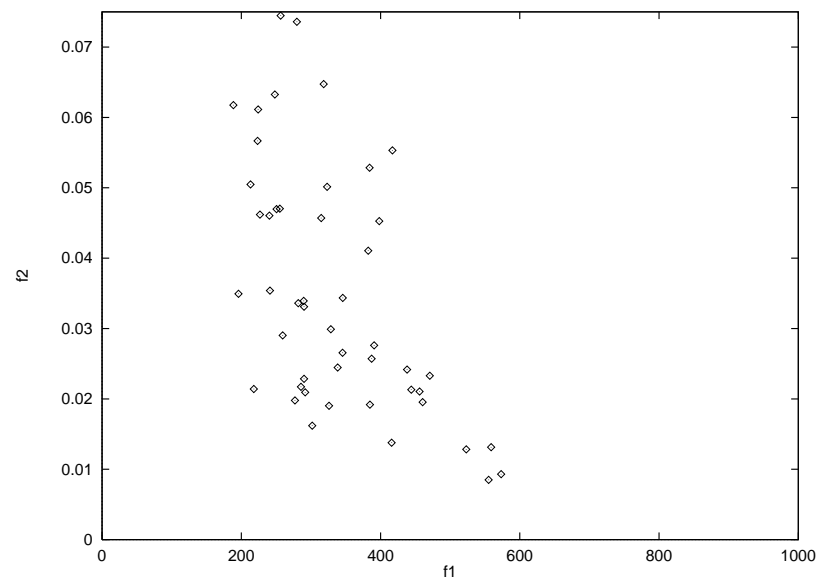


Figure 6.2: Example 1: Initial feasible region for Monte Carlo method.

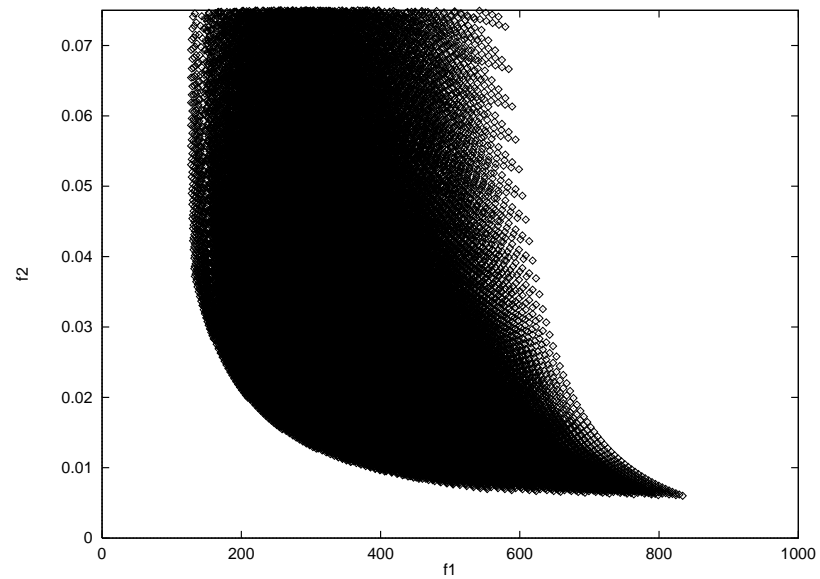


Figure 6.3: Example 1: Initial feasible region.

in the objective function space for this problem. It can be easily seen that the curve at the bottom of the graph is the Pareto front, since this is a minimization problem and the best compromises will be precisely at that part of the curve.

Let us start our analysis by presenting a methodology to compute the ideal vector, so that we can measure the quality of our solution. After that, I will show graphically the Pareto front of the problem, and the approximations produced with each method.

The ideal vector of this problem was computed using **Monte Carlo Methods 1 and 2** (generating 100 points) presented in Chapter 4, **Osyczka's multi-objective optimization system** and a GA (with a population of 100 chromosomes running during 50 generations) using binary and floating point representation, with the procedure described in Chapter 4 to adjust its parameters. The corresponding results are shown in Table 6.1, including the best results reported in the literature [1]. The results for Monte Carlo Method 2 are the same as for Method 1, and the results presented for the Min-max method are also the basis for computing the best trade-off for all the methods in Osyczka's system.

Since the Monte Carlo Methods previously mentioned and Osyczka's multiobjective optimization system do not generate the Pareto front, I will compare them with the GA-based approaches only in terms of the best overall result found. Besides those results, it is interesting to observe the initial distribution of points randomly generated by the method (see Figure 6.2). I used the same random number generator for both Monte Carlo methods and the GA, but since most GA-based methods do not check feasibility, the corresponding graph shows fewer points. Nevertheless, as we will see next, the GA is able to generate the Pareto front in a single run, whereas most mathematical programming techniques require

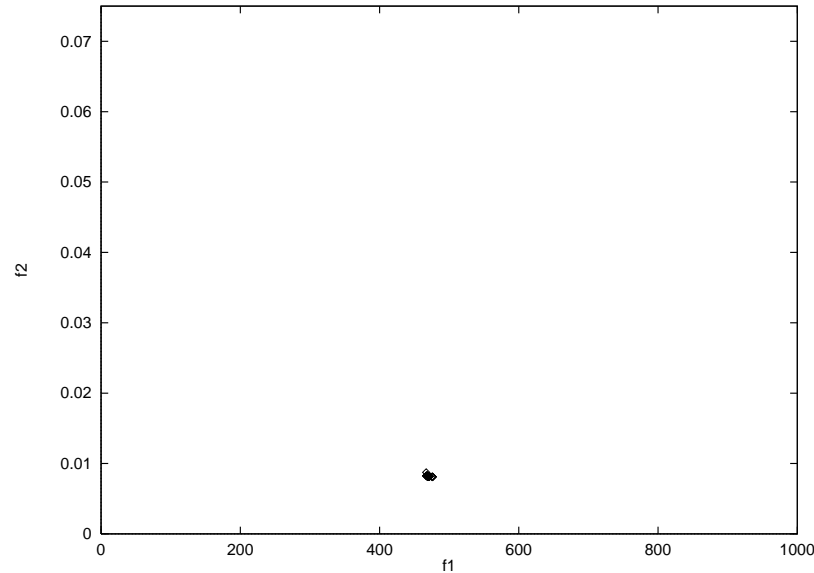


Figure 6.4: Example 1: The GA using a linear combination of the objectives with scaling, after 50 generations.

a lot of effort to do that, when possible, or simply generate a single final solution (the best overall).

We will start by analyzing the behavior of the **Lexicographic Method** at different stages of the search process. Figure 6.1 shows the distribution of points produced at generation zero using this method. Figure 6.38 shows the distribution of points at generation 20 using binary representation, and Figure 6.39 shows the corresponding distribution using floating point representation. It is easy to see how the points start grouping at the lower part of the Pareto front, but there are still several dominated points in the results. It is interesting to see how the floating point representation provides a more uniform distribution than binary representation, even when using the same method and the same parameters for the GA. Figure 6.39 shows how there are fewer dominated points at generation 20, and the Pareto front seems to be more clear. However, after 50 generations, binary representation seems to produce a better distribution (see Figure 6.40) than floating point representation (see Figure 6.41). There are also fewer points

displayed, because there is more redundancy of solutions using floating point representation. The reason for this behavior is that this representation scheme provides a faster convergence, and therefore the population at this stage of the search is starting to converge towards a global optimum. Notice, however, that when the GA uses floating point representation a better overall solution is found. I also compared the effect of a simple linear combination of objectives (addition or multiplication) using scaling, and the results after 50 generations are shown in Figure 6.4. Observe how the GA converges towards a global solution, and it finds a good trade-off, but it completely fails at delineating the Pareto front. Furthermore, the use of floating point representation produces a very good solution (better than the solution produced using binary representation), but the population totally converges to it after 50 generations. This shows the superiority of this scheme as a numerical optimization tool to achieve a single goal, but obviously a simple linear combination of objectives will not be suitable if we want to find the entire Pareto front or at least a good part of it. However, a linear combination of objectives is a good competitor for mathematical programming techniques, at least in this problem, since it works equally fast and it produces better results. In both previous cases a penalty function was used to “punish” the fitness each time that a constraint was violated, as reported in previous work [198] [199] [200].

Schaffer’s **VEGA** produces a rough approximation of a part of the Pareto front after 20 generations using binary representation (see Figure 6.5), but it still has several dominated points within the population. After 50 generations, however, the contour seems very well defined, even though there are still a few dominated points gathering inside the feasible zone (see Figure 6.6). The well-known “speciation” problem does not seem to be a big issue up to this point, because the method can generate in a reasonable amount of time most of the Pareto front,



Method	$x_1$	$x_2$	$x_3$	$x_4$	$f_1$	$f_2$	$L_p(f)$
Ideal Vector					127.46	0.0059	0.000000
Monte Carlo 1	77.57	20.59	3.47	3.88	401.77	0.0159	3.837581
Monte Carlo 2	75.01	31.02	1.76	2.48	277.09	0.0198	3.525181
Min-max (OS)	75.06	44.99	1.99	1.99	320.55	0.0167	3.350946
GCM (OS)	75.06	44.99	1.99	1.99	320.55	0.0167	3.350946
WMM (OS)	75.06	44.99	1.99	1.99	320.55	0.0167	3.350946
PMM (OS)	74.97	44.97	1.97	1.97	316.85	0.0170	3.360191
NMM (OS)	74.99	44.99	1.99	2.06	326.49	0.0164	3.332501
GALC (B)	80.00	50.00	0.92	3.98	463.99	0.0083	3.042494
GALC (FP)	80.00	50.00	0.90	3.80	445.55	0.0086	2.953417
Lexicographic (B)	80.00	45.25	0.98	2.73	319.95	0.0124	2.614093
Lexicographic (FP)	80.00	50.00	0.90	2.26	293.74	0.0134	2.572228
VEGA (B)	80.00	50.00	0.94	2.24	295.59	0.0134	2.589958
VEGA (FP)	80.00	23.33	3.52	5.00	479.49	0.0116	3.736026
NSGA (B)	80.00	44.28	2.39	4.35	555.19	0.0080	3.714160
NSGA (FP)	80.00	50.00	5.00	1.18	506.56	0.0132	4.210141
MOGA (B)	80.00	46.48	1.29	2.70	347.27	0.0119	2.743380
MOGA (FP)	80.00	30.38	0.90	3.53	279.95	0.0146	2.668388
NPGA (B)	78.75	36.69	1.40	3.71	372.17	0.0117	2.909137
NPGA (FP)	78.52	29.36	2.51	2.74	344.44	0.0160	3.409632
Hajela (B)	80.00	50.00	0.90	4.72	535.48	0.0072	3.418376
Hajela (FP)	80.00	50.00	1.92	5.00	634.05	0.0066	4.090622
GAminmax1 (B)	80.00	40.58	0.92	3.02	312.77	0.0127	2.603628
GAminmax1 (FP)	80.00	50.00	0.90	2.43	310.33	0.0126	2.568096
GAminmax2 (B)	80.00	35.06	0.91	3.57	316.20	0.0127	2.626021
GAminmax2 (FP)	80.00	39.32	0.90	2.85	290.92	0.0137	2.607544

Table 6.2: Comparison of the best overall solution found by each one of the methods included in MOSES for the first example. GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.5 (equal weight for both objectives).

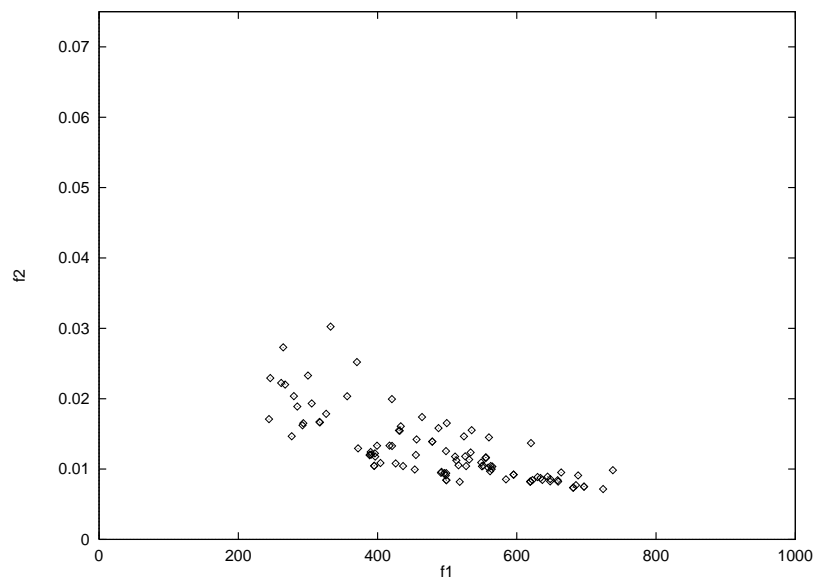


Figure 6.5: Example 1: Distribution of points using VEGA with a binary representation at generation twenty.

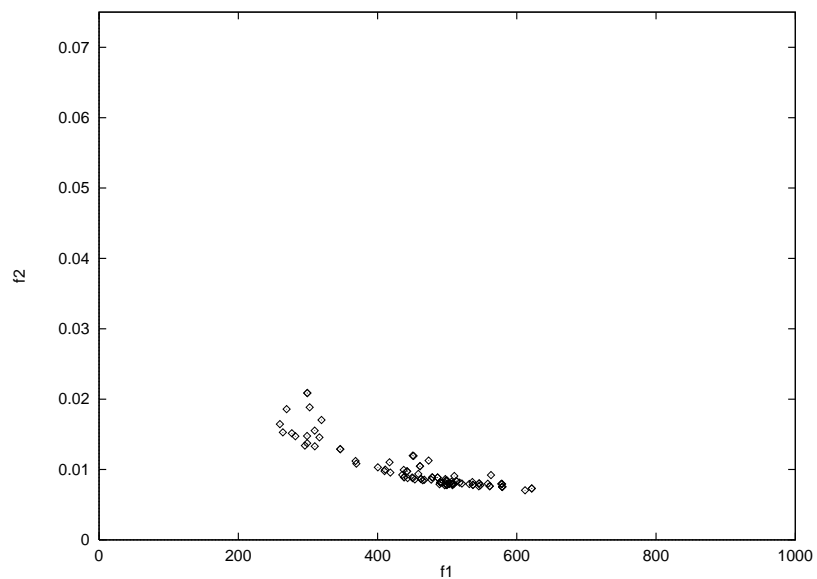


Figure 6.6: Example 1: Distribution of points using VEGA with a binary representation at generation fifty.

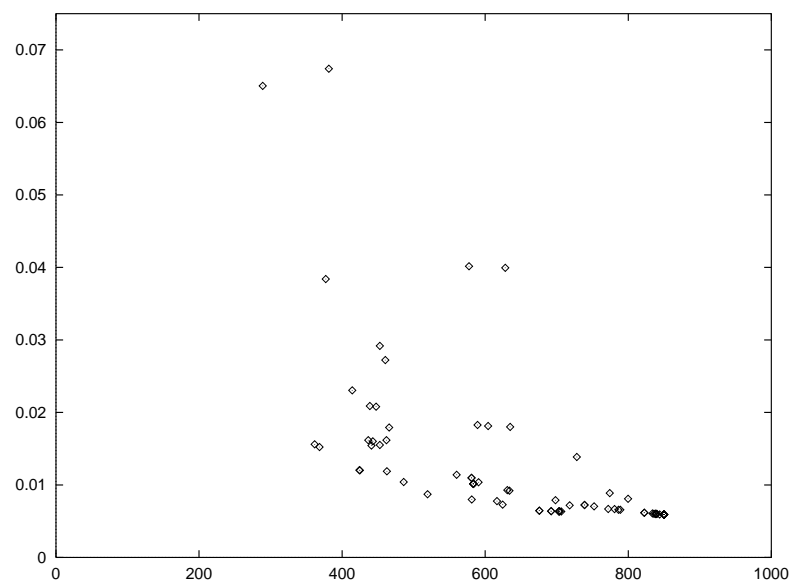


Figure 6.7: Example 1: Distribution of points using VEGA with floating point representation at generation twenty.

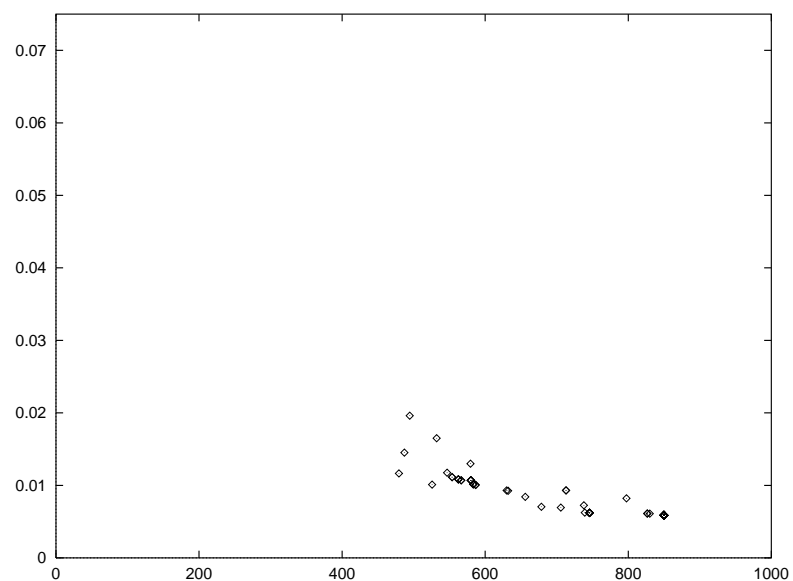


Figure 6.8: Example 1: Distribution of points using VEGA with floating point representation at generation fifty.

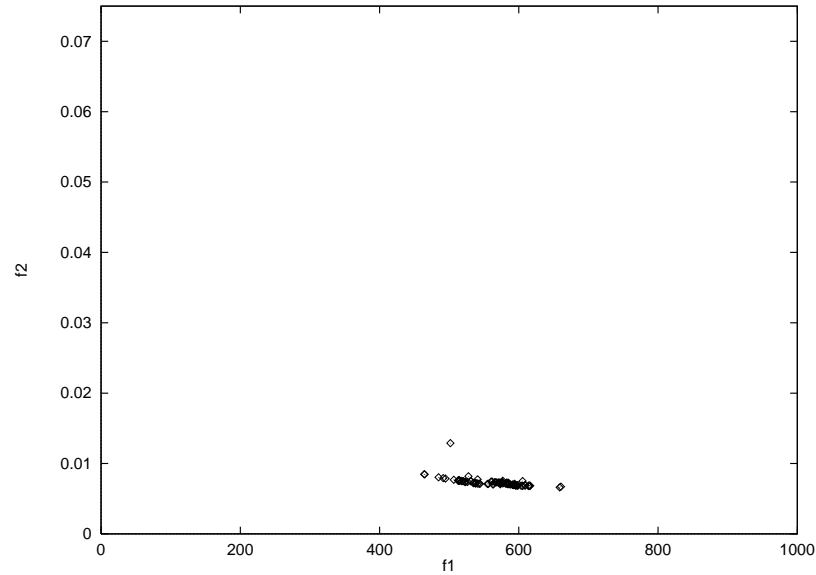


Figure 6.9: Example 1: Distribution of points using VEGA with binary representation at generation 100.

and the best overall solution that it produces is also very good. Floating point representation does not help very much in this example, since it produces very sparse distributions, as can be seen in Figure 6.7. Even when the Pareto front is partially visible, there are many dominated individuals present in the population. After 50 generations, the quick convergence property of this scheme representation newly produces a stable population that partially converges towards the appropriate Pareto contour, but the set is not as good as the one produced with binary representation, since the visible part of the Pareto front is smaller in this case (see Figure 6.8). Interestingly, against what one could think, the best overall solution turned out to be poorer in this case than when using binary representation. The reason is that the population started converging towards a Pareto solution at the bottom of the Pareto front, instead of converging towards the left angle of the contour as most of the previous techniques. It should be mentioned that, as Srinivas and Deb indicate [181], after many generations, VEGA converges to very few non-dominated solutions, as can be seen in Figure 6.9. In fact, after 500

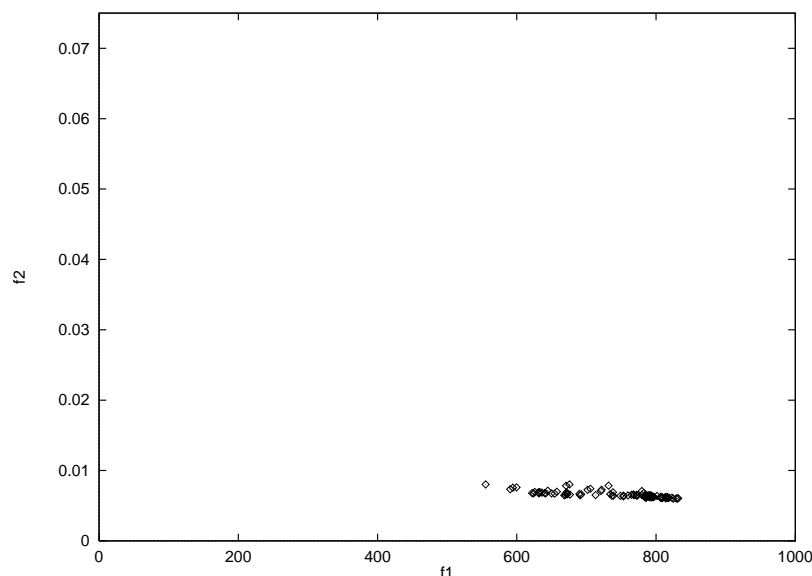


Figure 6.10: Example 1: Distribution of points using NSGA with binary representation at generation fifty.

generations, it converges to only 2 non-dominated solutions. As I said before, the accelerated convergence rate provided by the floating point representation shows this undesirable property of the algorithm very early in the process. VEGA is not well suited for problems with concave trade-offs, as Fonseca and Fleming indicate [188], because each objective is weighted proportionally to the size of each subpopulation and to the inverse of the average fitness (in terms of that objective) of the whole population at each generation if proportionate selection is used. Richardson et al. [172] noted that shuffling and merging all subpopulations corresponds to averaging the *normalized* fitness components associated with each of the objectives. Different non-dominated individuals will be assigned different fitness values, in a direct contradiction of the definition of non-dominance. This method clearly seeks for individuals that excel in one objective and not for the best trade-offs. However, in problems in which the trade-off surface is convex (like this example), it is possible to get a good approximation of the Pareto front after a relatively small number of generations.

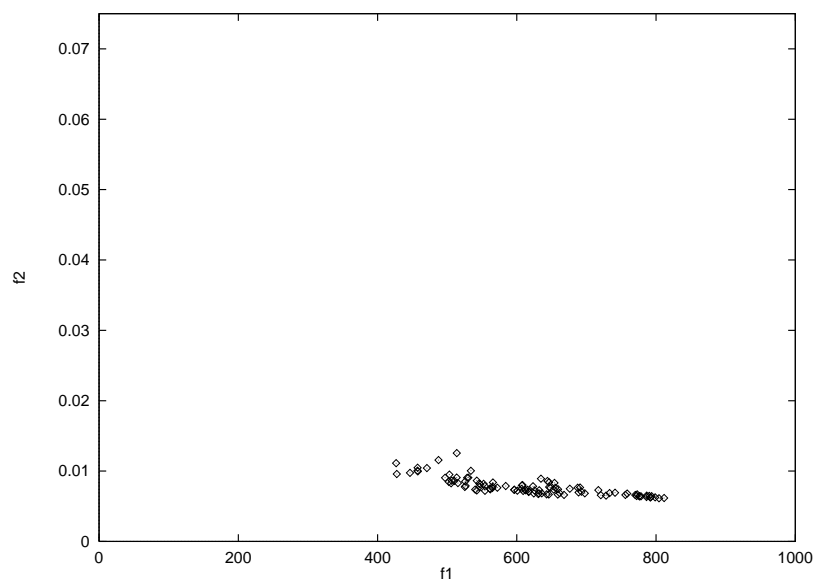


Figure 6.11: Example 1: Distribution of points using NSGA with binary representation at generation twenty.

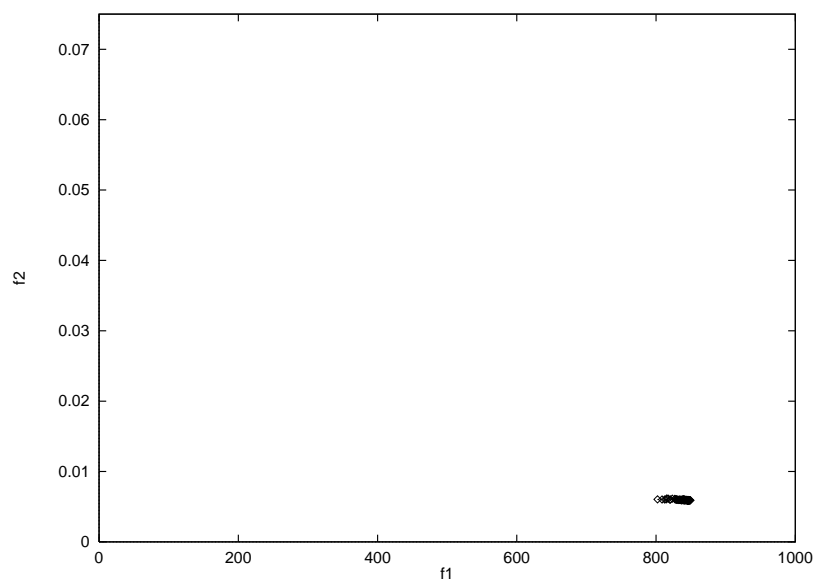


Figure 6.12: Example 1: Distribution of points using NSGA with binary representation at generation 500.

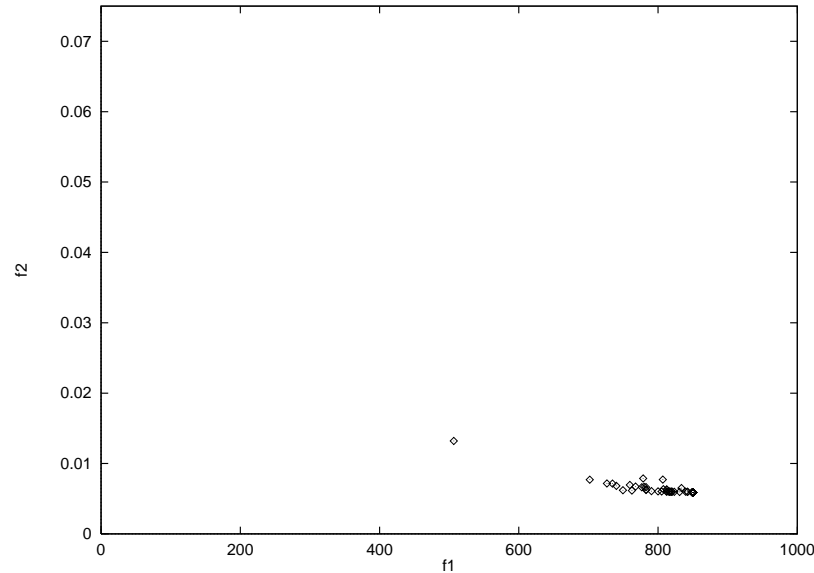


Figure 6.13: Example 1: Distribution of points using NSGA with floating point representation at generation 50.

My version of Srinivas' **NSGA** produces a very smooth contour of the Pareto front using binary representation after 50 generations, as can be seen in Figure 6.10. Even after only 20 generations, the result is very good (see Figure 6.11). Contrary to what Srinivas and Deb [181] indicate, this technique does not seem as stable as they claim, since after 500 generations the population has almost converged to a very small subset of the Pareto set (see Figure 6.12). This effect can be anticipated again by using floating point representation with only 50 generations (see Figure 6.13). Furthermore, this technique also produces only a fraction of the Pareto front since the beginning, because non-dominated solutions placed at the bottom of the Pareto-front seem to guide the search in this example. This technique requires much more CPU time than any of the other previous techniques, and due to the fact that it does not manipulate at any moment the real fitnesses of the objective functions, the best overall solution produced at the end is very poor for both representation schemes, but worse for floating point representation because of its trend to stabilize the population around a single non-dominated

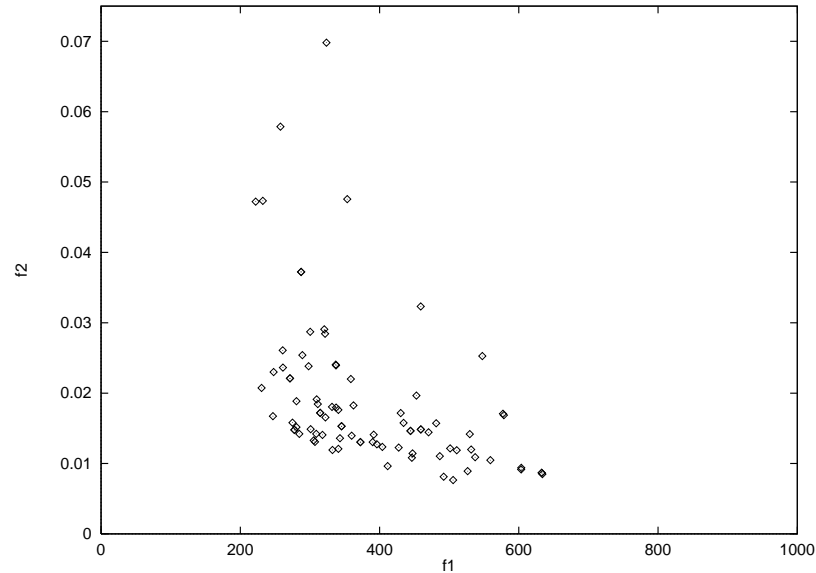


Figure 6.14: Example 1: Distribution of points using MOGA with binary representation at generation 20.

solution which in this case is not a very good trade-off (see Table 6.2). It should be also pointed out that in the original definition of the algorithm constraints were not taken into consideration, and they have to be taken care of separately, or infeasible solutions could be generated and even dominate the search. It has been reported by the authors of this algorithm [181] that it is highly susceptible to the value of  $\sigma_{share}$ , and they recommend to follow the guidelines given by Deb and Goldberg [201]. In this example, however, there was no notorious change in the distribution of points for different values of  $\sigma_{share}$ . The identification of non-dominated individuals and the determination of niche sizes are processes  $O(n^2)$ . Additionally, since the authors use stochastic remainder selection, the algorithm could become even slower.

Fonseca's MOGA works in a very interesting manner. First, when I ran it with a binary representation scheme during 20 generations, a promising contour appeared (see Figure 6.14). However, after 50 generations, even when the contour is better than that produced by NSGA, there is still a relatively spread



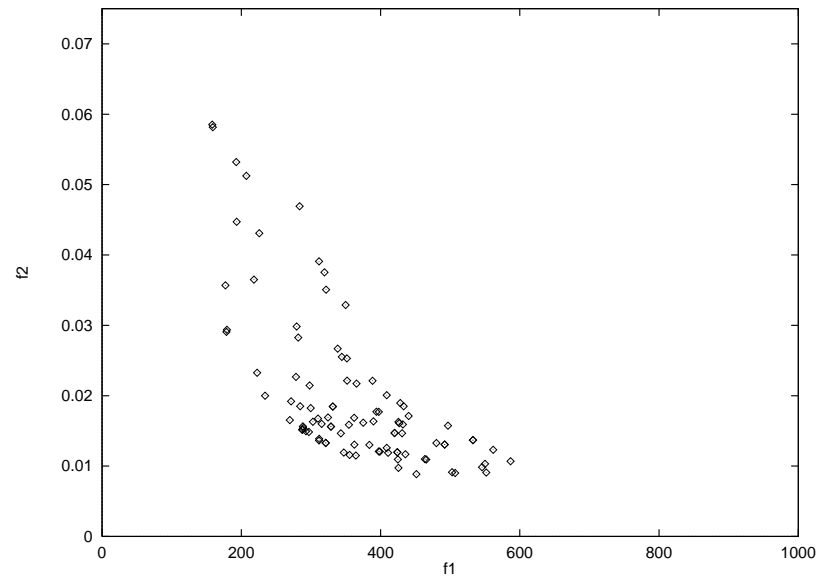


Figure 6.15: Example 1: Distribution of points using MOGA with binary representation at generation 50.

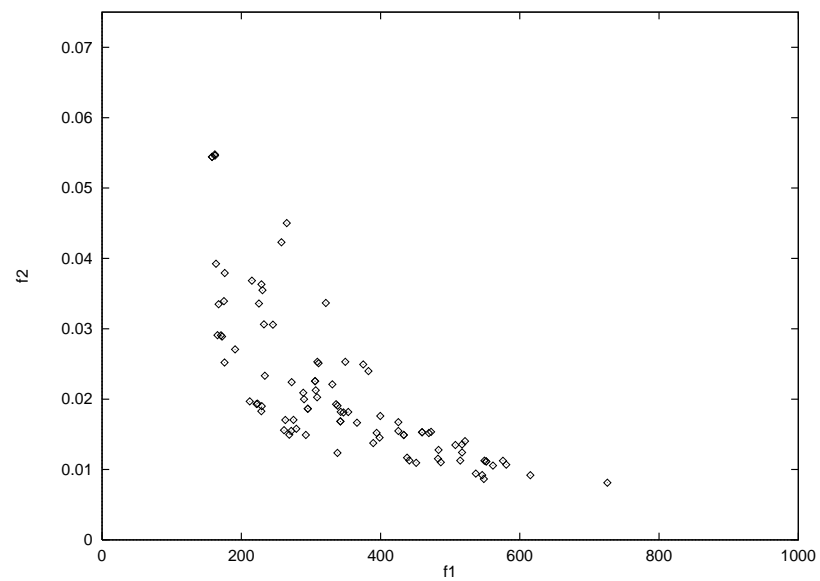


Figure 6.16: Example 1: Distribution of points using MOGA with binary representation at generation 100.

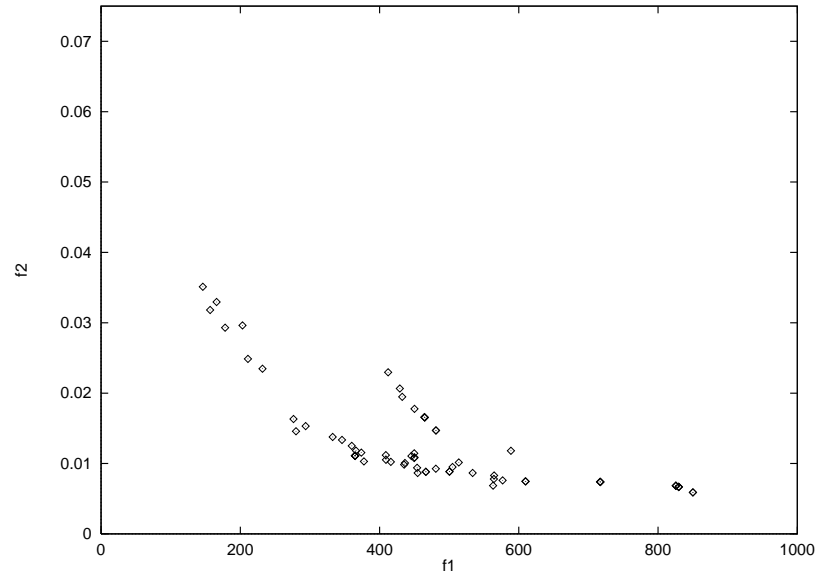


Figure 6.17: Example 1: Distribution of points using MOGA with floating point representation at generation 50.

population (see Figure 6.15). I increased the number of generations, and after 100 generations, the front is completely visible, but there are still a lot of dominated points in the graph (see Figure 6.16). As I kept increasing the number of generations, the technique started behaving as NSGA, and after 500 generations only 3 non-dominated points remained alive, with an almost total convergence of the population toward one of them. The best overall solution produced by the method is better than the result produced by a linear combination of objectives, but not as good as the solution provided by the Lexicographic method. This should not surprise us, because this algorithm uses the true objective function values to guide the search, but the niching technique used avoids to find the best overall individual in the population. Again, floating point representation produces a better overall result, but because of its accelerated convergence, it produces a fuzzier Pareto front (see Figure 6.17). I also used a value of  $\sigma_{share} = 0.1$  as in NSGA, and changing this value didn't affect the result produced by the algorithm. To compute the niche size I used the method suggested by Fonseca and Fleming [180].

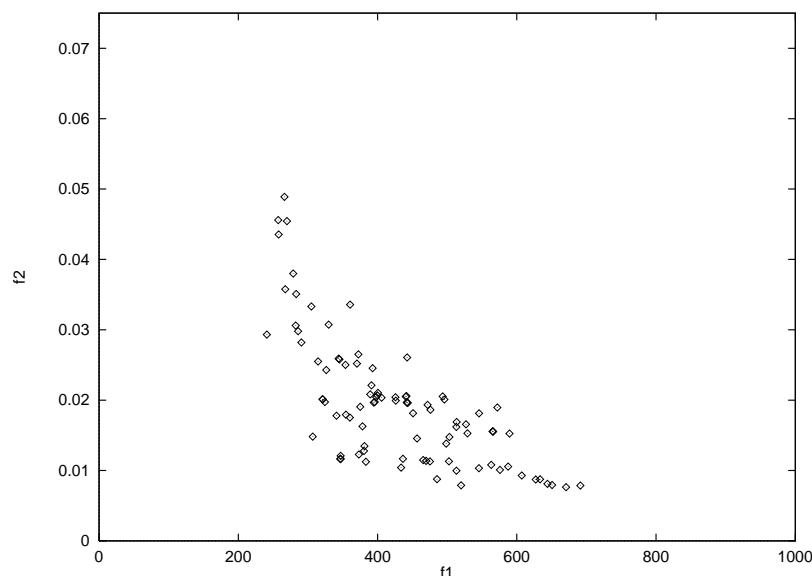


Figure 6.18: Example 1: Distribution of points using NPGA with binary representation at generation 20.

This algorithm is more efficient than NSGA, but its behavior does not seem very appropriate, because even when the population is diversified after a few generations, there are still a lot of non-dominated points present in the population. The main criticism to this technique is the fact that it does sharing on the objective function values to distribute the population over the Pareto front, instead of doing it on the parameter values as NSGA [181]. This makes the algorithm incapable of finding multiple solutions in problems where different Pareto optimum points correspond to the same objective function value. Finally, as in the case of NSGA, MOGA does not check feasibility, and consequently is possible to have infeasible solutions at late stages of the search. This is an important drawback, because one of this solutions could dominate the others and then the GA would converge to an infeasible region. It is therefore the responsibility of the user to check feasibility of the solution at some point, and probably penalize those solutions even if they apparently dominate others.

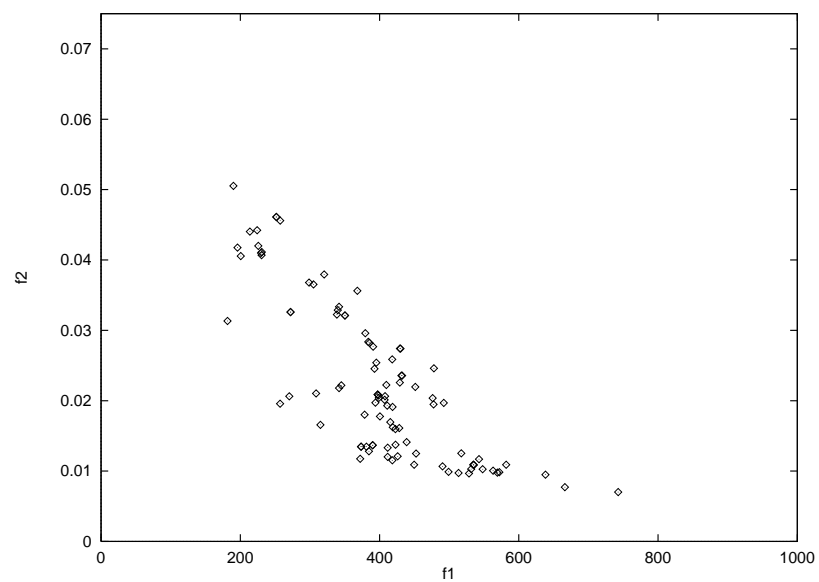


Figure 6.19: Example 1: Distribution of points using NPGA with binary representation at generation 50.

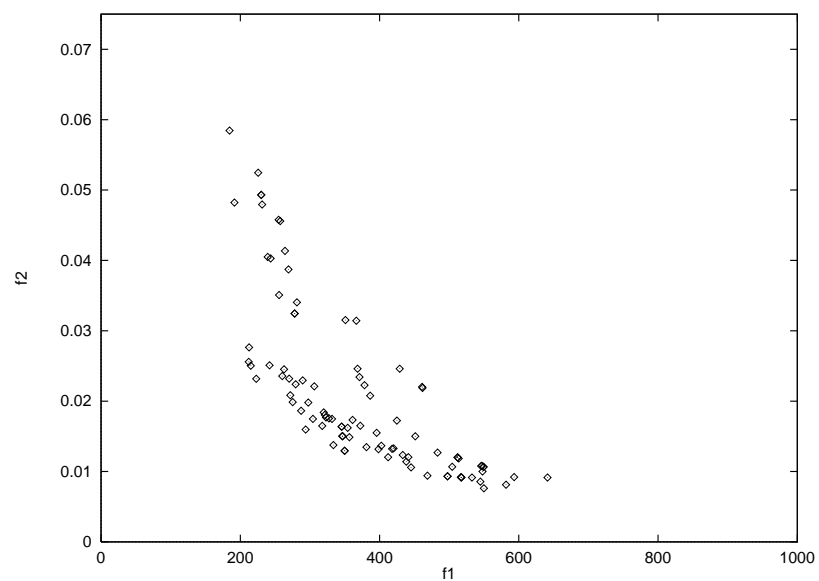


Figure 6.20: Example 1: Distribution of points using NPGA with binary representation at generation 100.

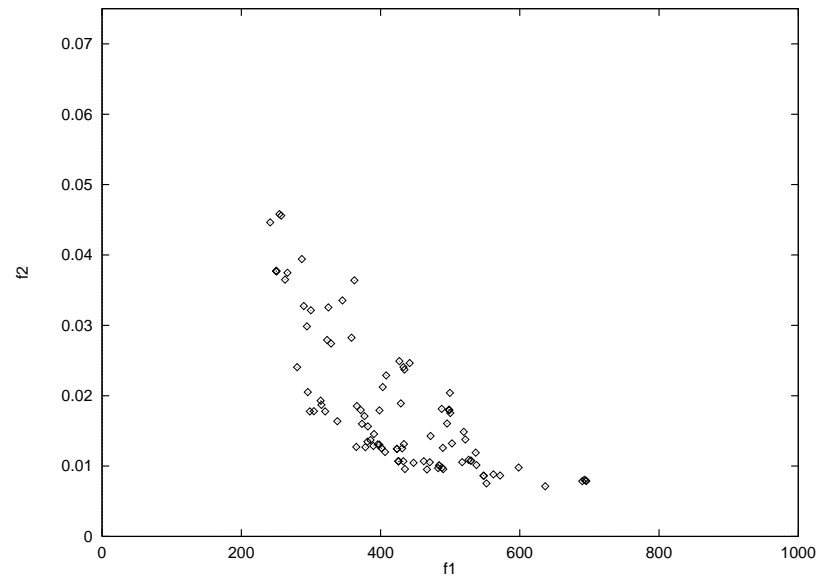


Figure 6.21: Example 1: Distribution of points using NPGA with binary representation at generation 500.

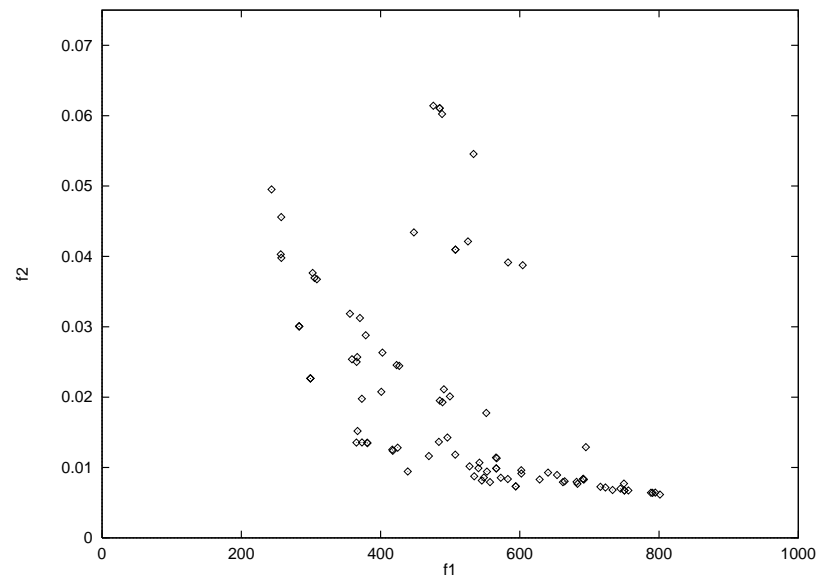


Figure 6.22: Example 1: Distribution of points using NPGA with binary representation at generation 50 with  $\sigma_{share} = 1.0$ .

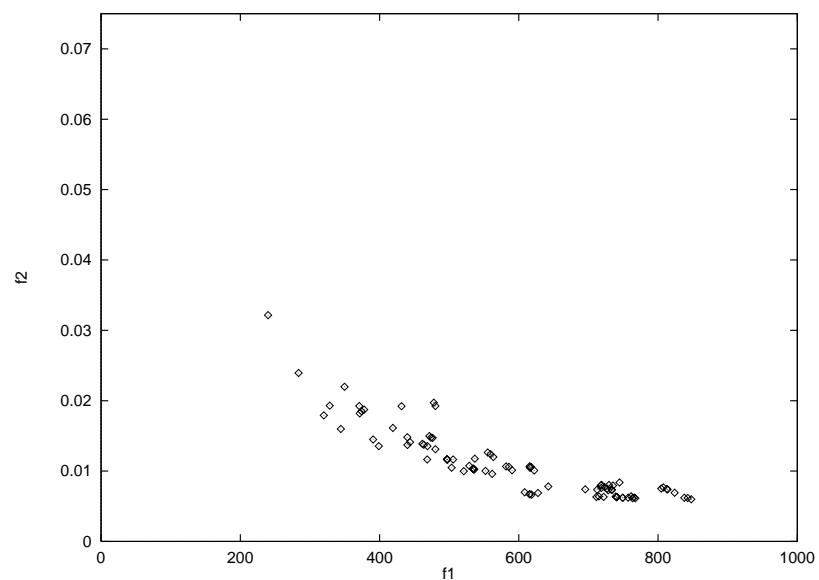


Figure 6.23: Example 1: Distribution of points using NPGA with floating point representation at generation 50.

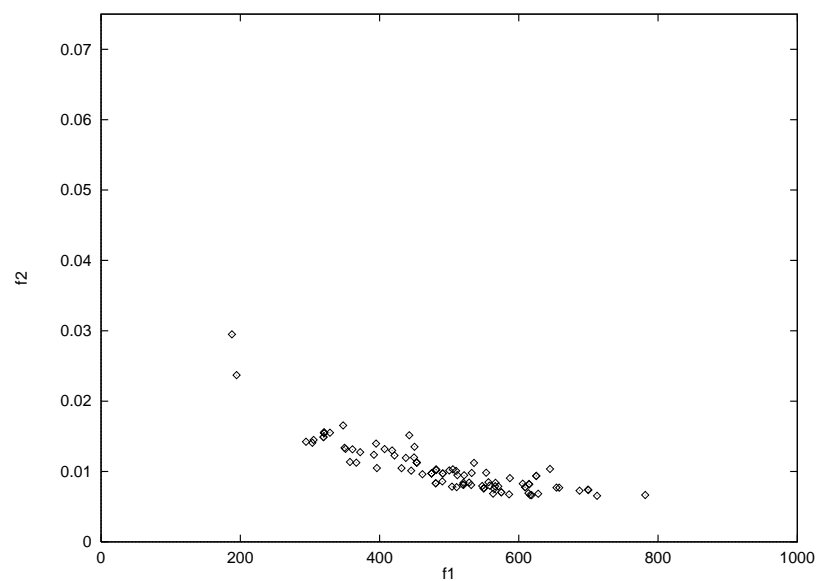


Figure 6.24: Example 1: Distribution of points using NPGA with floating point representation at generation 100.

When I started using the NPGA developed by Horn and Nafpliotis [183] my first impression is that its behavior was similar to MOGA, but that was not the case. At an early stage in the search process, the algorithm produces a very smooth distribution, even though there are still many dominated points in the population (see Figure 6.18). After 50 generations, the behavior of this technique looks similar to MOGA, because the population seems too sparse, with many dominated individuals (see Figure 6.19). However, if we continue up to 100 generations, the front is more stable and there is no convergence to a single point (see Figure 6.20). Remarkably, after 500 generations, the population looks very well distributed, proving the stability of the algorithm (see Figure 6.21). Furthermore, this technique is able to display a larger section of the Pareto front than the other GA techniques based on Pareto-ranking selection. Considering the scarce manipulation of the fitness function, this technique produces a reasonably good best overall solution. In this case, the value of  $\sigma_{share}$  plays a critical role, and if we increase it from 0.1 (same value used in the previous techniques) to 1.0, we will see that the technique has more difficulties to form niches (see Figure 6.22). Furthermore, the main criticism of this technique is that it requires an additional parameter to be adjusted: the size of the tournament  $t_{dom}$  [183]. As indicated by Srinivas and Deb [181], if this value is too small, some non-dominated points will not be found, and if it is too large, we could have premature convergence. I used  $t_{dom} = 10$  to generate the previous graphs, but modifying this parameter I ran into the problems mentioned by Srinivas and Deb. In terms of representation schemes, a floating point representation produced a very smooth surface in only 50 generations (see Figure 6.23), but the best overall solution was poorer than the value produced using binary representation. The reason for this is that the algorithm converged toward the bottom part of the Pareto front, as in the case of

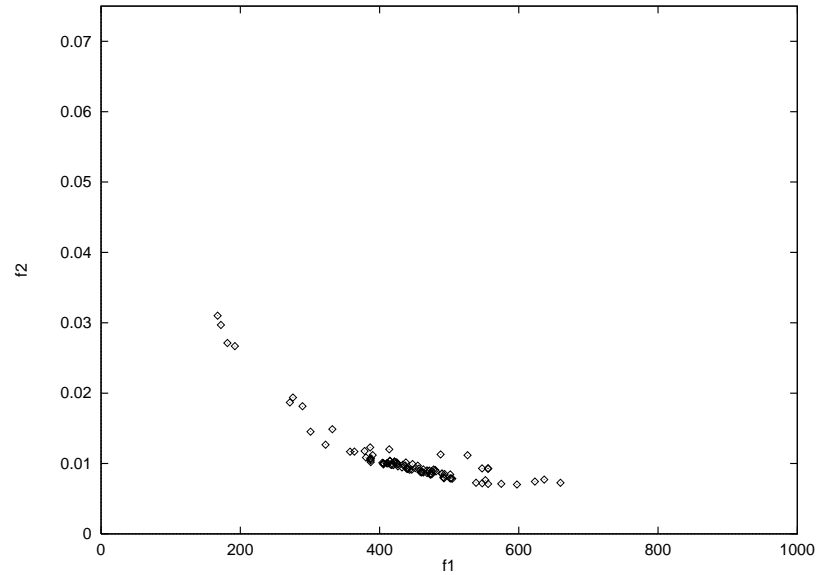


Figure 6.25: Example 1: Distribution of points using Hajela's method with binary representation at generation 20.

NSGA and MOGA, instead of moving towards the left side of the contour in which the best overall non-dominated individuals reside. Nevertheless, the technique shows the same stability after 100 generations using this representation scheme (see Figure 6.24) as a good indication of its strength. The only thing that prevents me from recommending this representation to accelerate convergence in this case, is that when floating point representation is used, the algorithm becomes more sensitive to the parameters, so that if  $\sigma_{share}$  and/or  $t_{dom}$  is slightly modified, the behavior of the technique abruptly changes. In that direction, Horn and Nafpliotis [183] give some guidelines derived from their empirical study of their algorithm to choose these values.

As could be expected from a Non-Pareto approach, the performance of Hajela's method was not very stable with this example. This weighted min-max approach produces a very good distribution of points over the Pareto front after 20 generations (see Figure 6.25). At that stage of the search, the performance of the algorithm seems excellent, since the contour seems very clear, and the best



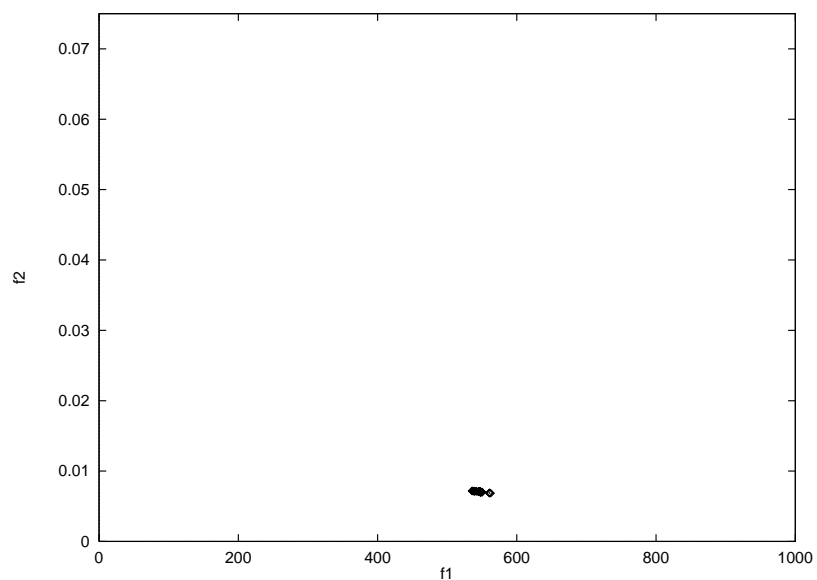


Figure 6.26: Example 1: Distribution of points using Hajela's method with binary representation at generation 50.

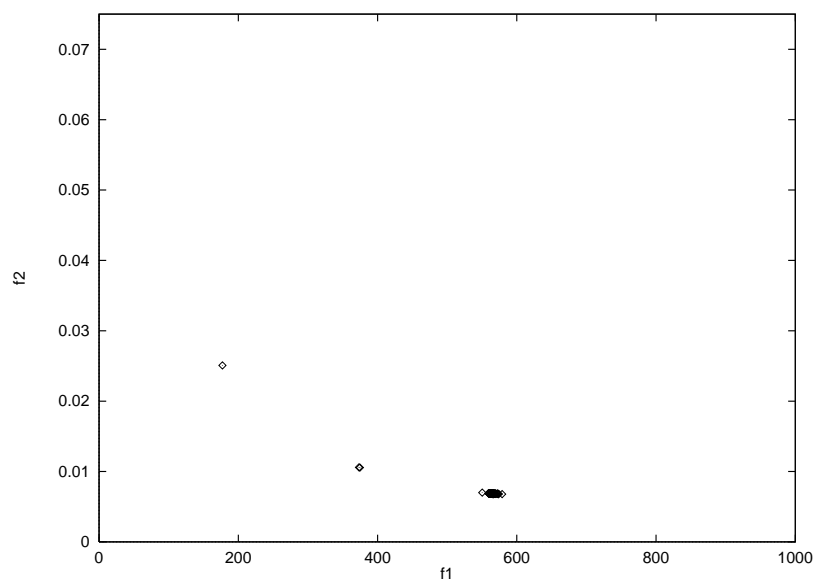


Figure 6.27: Example 1: Distribution of points using Hajela's method with binary representation at generation 50 using 500 individuals.

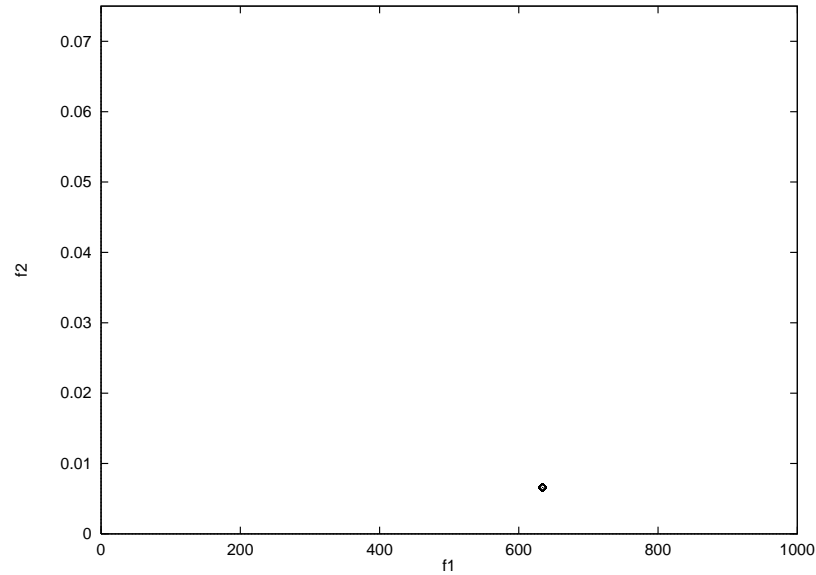


Figure 6.28: Example 1: Distribution of points using Hajela's method with floating point representation at generation 50.

overall individual found is very good ( $L_p = 2.676497$ ). Unfortunately, when the number of generations increases, the technique quickly converges towards a single point, which happens to be a poor non-dominated solution (see Figure 6.26). A reasonable assumption would be to attribute the failure of the technique to the use of a small population, but that is not the case, because even using a population of 500 chromosomes, only two extra non-dominated points appeared, as shown in Figure 6.27). The main drawback of this technique seems to be its lack of stability, because it loses its smooth initial distribution very easily, besides requiring much extra information, such as a list of possible weights,  $\sigma_{share}$  (chosen to be 0.1 in this case, as in the previous techniques), and the mating restriction threshold (chosen to be 0.1 in this case). In this particular example, the increment and decrement of  $\sigma_{share}$  and the mating threshold didn't affect the solution in any significant way. In terms of representation schemes, the use of a floating point representation only makes things worse, because the entire population converges, after only 50 generations to a unique solution, which turns out to be worse than the best

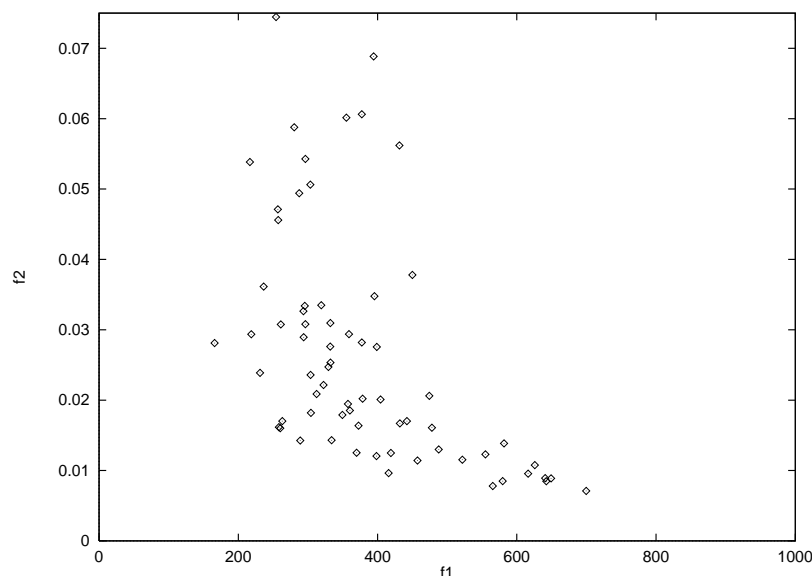


Figure 6.29: Example 1: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero.

trade-off produced when using a binary representations scheme (see Figure 6.28). In fact, after only 10 generations, the GA converges to such solution, showing the premature convergence problem of this approach. Also, as in the previous techniques discussed, this algorithm does not check feasibility of the solutions at any time, so it could be the case that after several generations a bad solution dominates the others and the GA could converge towards a invalid solution.

Before showing the results produced with my method based on the min-max approach, I want to show the distribution of points at generation zero, since in this case the algorithm ensures that only feasible solutions are generated. It can be seen in Figure 6.29 that the initial distribution has a fairly large amount of points near the Pareto front, but most of them are sparsely distributed. However, if we constraint the initial generation to be totally feasible, and if we also restrict the operators to produce only feasible children, then we can ensure, at each moment, that the points encoded are in the feasible zone. Now, in terms of the results, this method produced a very uniform distribution of points over the Pareto front,

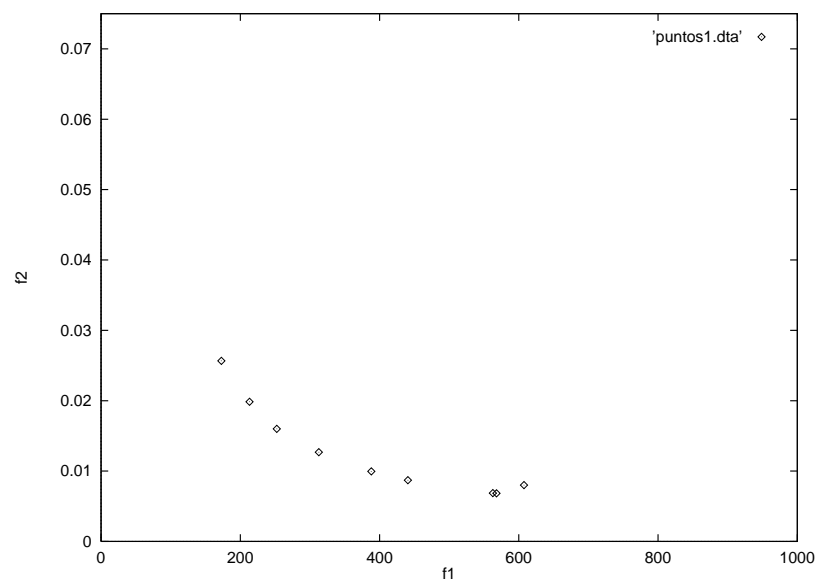


Figure 6.30: Example 1: Distribution of points using my method based on the min-max algorithm with binary representation at generation 50.

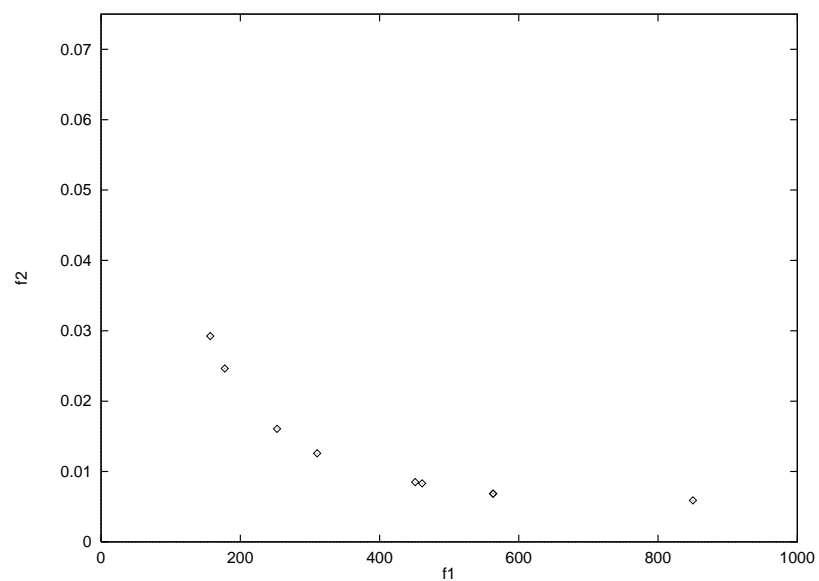


Figure 6.31: Example 1: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 50.

showing the most part of it after 50 generations (see Figure 6.30) using binary representation. The best overall individual is also the best of all (only surpassed by VEGA) using binary representation (see Table 6.2). The main drawbacks of this approach is that an independent run is required for each weight combination, so if there is no access to a parallel architecture, this will consume much more time than with previous approaches. Additionally, the set of weight combinations should be provided by the user, and the ideal vector must be known. In this last point, however, I should point out that another goal of this work is to show that the GA can be used as an effective single-objective numerical optimization tool. For that sake, I developed the methodology described in Chapter 4 to adjust the parameters of the GA, and as the results indicate, the GA can find better answers than mathematical programming techniques using this dynamic adjustment of parameters. The trade-off in that respect is either to run the GA 10 or 20 times without any clue of what the most appropriate parameters are, or to run it 81 times with this methodology and get at least a sub-optimal solution. More on this parameter adjustment scheme will be discussed in a later section. On the other hand, my method guarantees that only feasible solutions are generated at all times, because it imposes restrictions on the production of chromosomes at generation zero, and during crossover and mutation. Also, it is very stable, because at all times only single objective optimization problems are being solved, and no niching or mating parameters of any sort are required. When using this technique with a floating point representation, the Pareto front generated is slightly less uniformly distributed, but a larger segment of the contour is visible after 50 generations (see Figure 6.31). Additionally, the best overall solution is the best trade-off that I found for this problem. This corroborates once more the efficiency of floating point representation in numerical optimization problems like this.

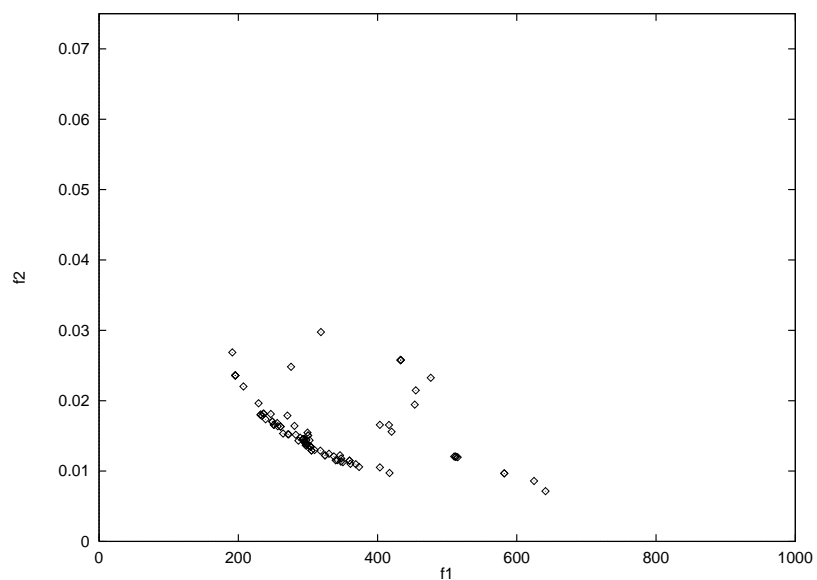


Figure 6.32: Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20.

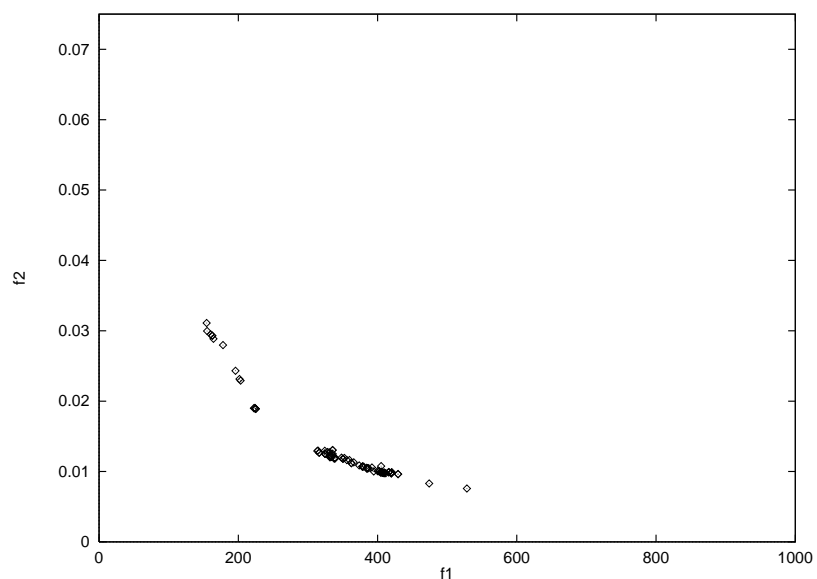


Figure 6.33: Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 50.

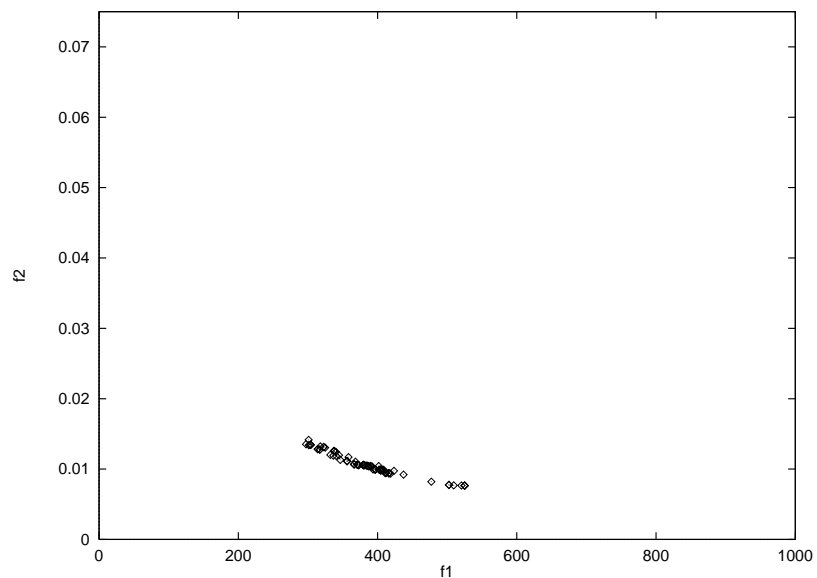


Figure 6.34: Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100.

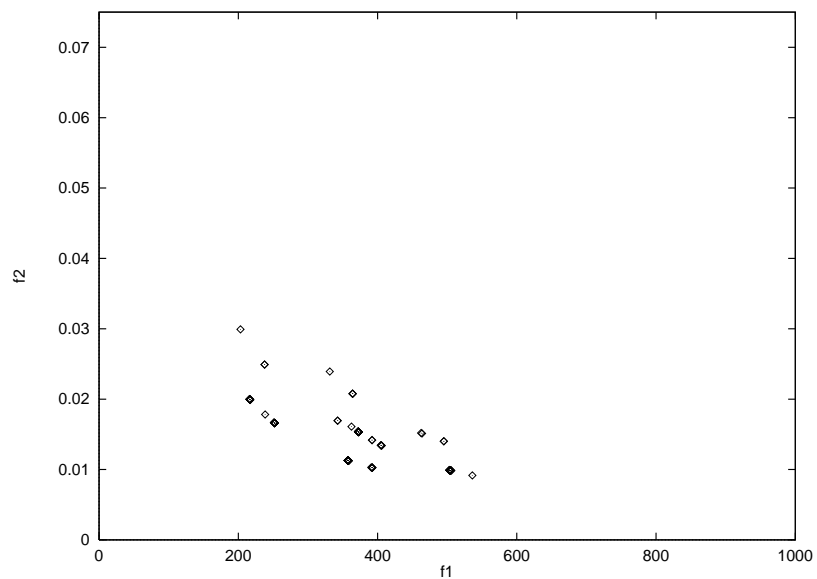


Figure 6.35: Example 1: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 500.

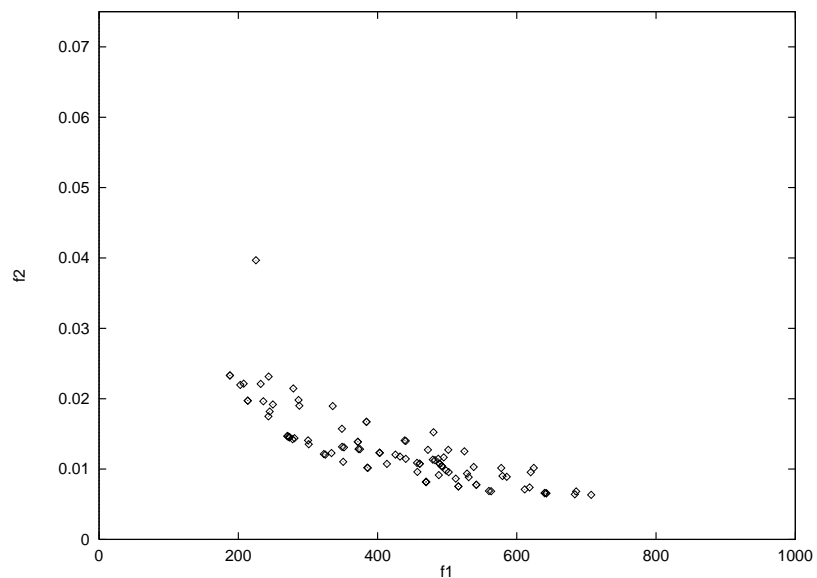


Figure 6.36: Example 1: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20.

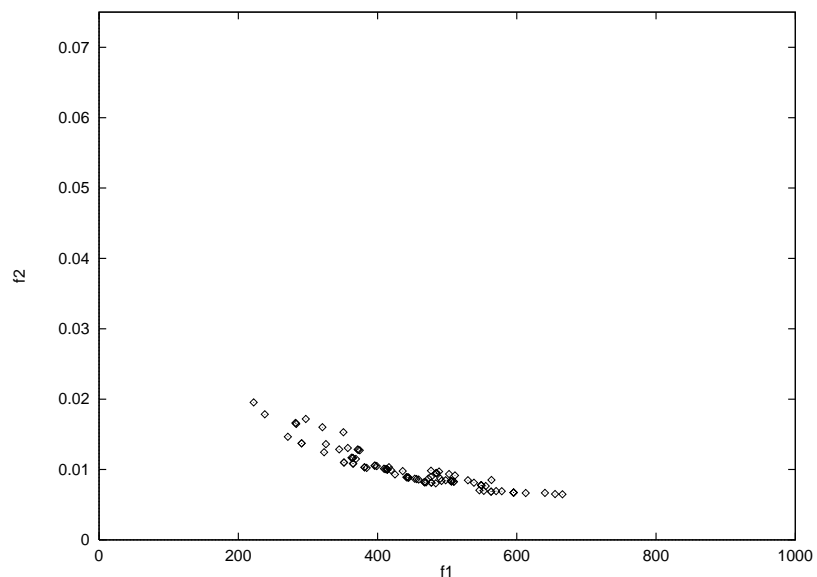


Figure 6.37: Example 1: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 50.



Finally I tried my min-max selection strategy with sharing. Using the same initial population shown in Figure 6.29, this algorithm has a very stable behavior if a good value for  $\sigma_{share}$  is used (I chose 0.5 for this example). After 20 generations using binary representation, the algorithm starts producing a good portion of the Pareto front, but there are still several dominated points present (see Figure 6.32). At generation 50, however, the contour of the Pareto front seems very clear (see Figure 6.33). At Generation 100, most points are grouping within the inferior part of the Pareto curve (see Figure 6.34), forming a line, but without converging to a unique solution. After 500 generations, the points seem to be spreading again, but a good portion of the Pareto front is still visible (see Figure 6.35), and the population has not converged to a single non-dominated point. The use of floating point representation produces better results in all aspects. After 20 generations, the population is spread over a larger area, covering the Pareto front completely (see Figure 6.36). After 50 generations, there is a clearer Pareto front than before, and the part containing the best non-dominated individuals is displayed (Figure 6.37) as the results from Table 6.2 reflects, since we were able to find a better overall result in this case. It should be mentioned that several of the previous GA-based techniques, such as MOGA and NSGA generated poorer overall solutions because they were only able to generate points at the right hand side of the solution space, instead of moving towards the curve of global optima. This did not happen with this approach, in which a more uniform distribution was achieved even after many generations.

Table 6.2 compares the best overall solution found by this and the remaining techniques using both representation schemes, with respect to the ideal vector. To evaluate these results, I used as a parameter the maximum deviation from the ideal vector, which is defined by

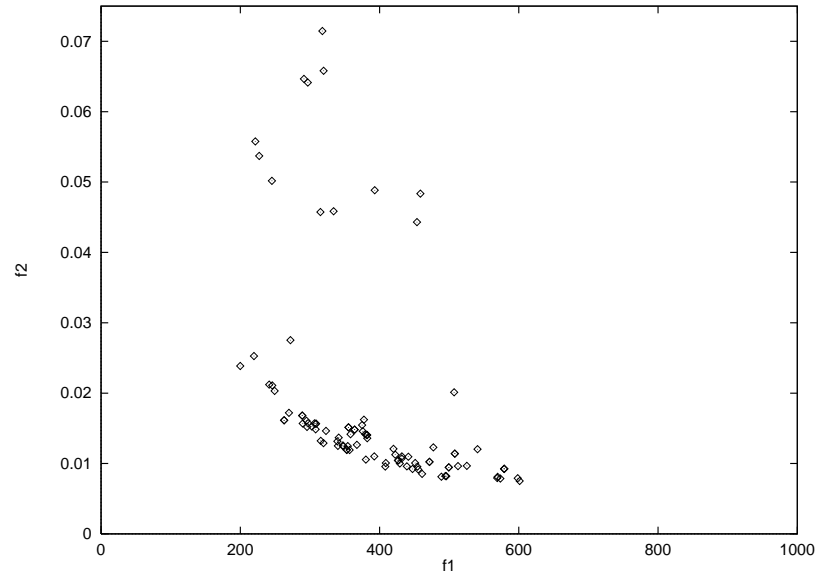


Figure 6.38: Example 1: Distribution of points using the Lexicographic Method with a binary representation at generation twenty.

$$L_p(f) = \sum_{i=1}^k \left| \frac{f_i^0 - f_i(x)}{\rho_i} \right| \quad (6.4)$$

where  $\rho_i = f_i^0$ , or  $f_i(x)$ , depending on which gives the maximum value for  $L_p(f)$ ,  $k$  is the number of objectives and  $f_i^0$  is the ideal vector.

Notice that Osyczka's Multiobjective Optimization System requires very good guesses to derive a good solution, since it normally moves within a very small window, and the final vector that it produces is always very close to the values given by the user. The reason is that the system was originally designed to be used interactively, so that the user could be guessing around what region he/she wanted to get a solution, and the program would sketch a solution (if any) around a certain vector of decision variables. Unfortunately, that is not very satisfactory in practice, because the user may completely ignore the shape of the feasible region and there could be a fairly large search space to trace.

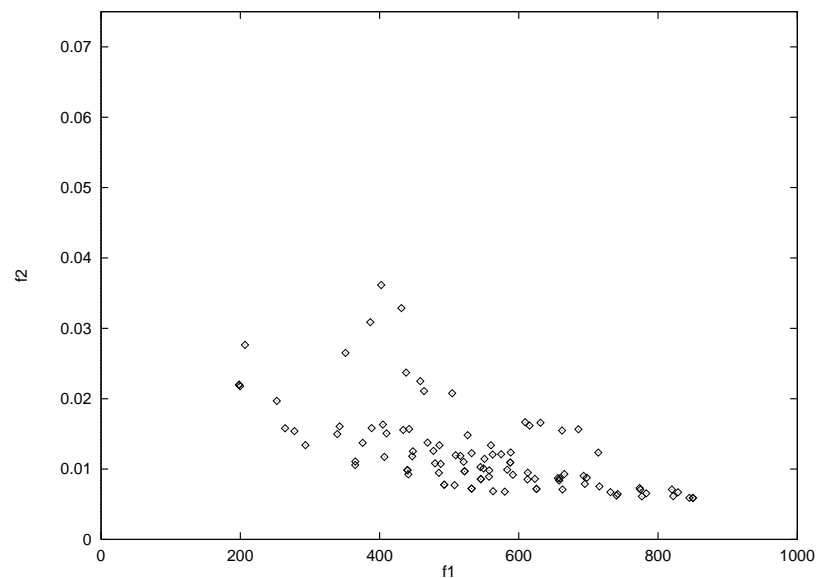


Figure 6.39: Example 1: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty.

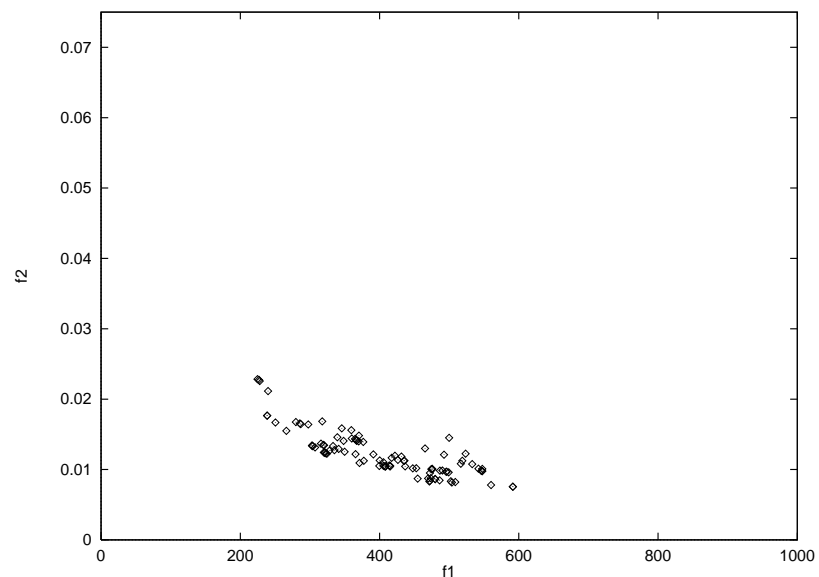


Figure 6.40: Example 1: Distribution of points using the Lexicographic Method with a binary representation at generation fifty.

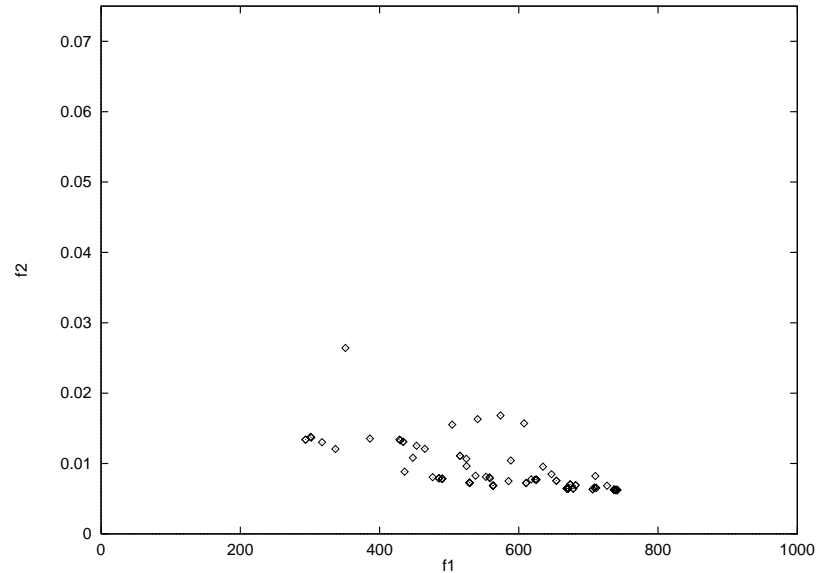


Figure 6.41: Example 1: Distribution of points using the Lexicographic Method with a floating point representation at generation fifty.

### 6.3 Example 2 : Machining Recommendations

In this example I will not show any graphical representation of the Pareto front, because there are too many objective functions, and plotting them is not possible. So, for this problem I will concentrate on how good all the techniques do in terms of getting the best overall result. First, I will start by showing the ideal vector, according to each one of the techniques used in the previous example. It is very interesting that this time the GA with floating point representation generated all but one element of the ideal vector. For the value of SI, the GA with binary representation found a slightly better value. It should be noted that in this problem, we are minimizing the first objective function, while maximizing the others. As can be seen from the results presented in Table 6.3, I found even better results than those reported in the literature.

Since I am only measuring the performance of each technique in terms of the best solution overall, there is not much to say about the results presented in Tables 6.4 and 6.5. One of the interesting things observed during the experiments

Method	$v$ (sfm)	$f$ (ipr)	$d$ (in)	SR ( $\mu$ in)	SI (% udmg)	TL (min)	MRR (in <sup>3</sup> /min)
Monte Carlo 1	1105.94	0.0020	0.0695	<b>14.03</b>	56.15	39.67	1.87
Monte Carlo 1	1105.94	0.0020	0.0695	14.03	<b>56.15</b>	39.67	1.87
Monte Carlo 1	1014.64	0.0026	0.0739	18.78	51.18	<b>40.76</b>	2.35
Monte Carlo 1	949.56	0.00330	0.0925	22.16	51.06	34.22	<b>3.48</b>
Min-Max (OS)	1200.00	0.0016	0.0496	<b>13.02</b>	54.18	51.14	1.18
Min-Max (OS)	1200.00	0.0016	0.0496	13.02	<b>54.18</b>	51.14	1.18
Min-Max (OS)	1200.00	0.0016	0.0496	13.02	54.18	<b>51.14</b>	1.18
Min-Max (OS)	1200.00	0.0026	0.0499	21.95	50.49	44.43	<b>1.86</b>
GA (Binary)	1045.64	0.0020	0.100	<b>11.31</b>	62.52	30.42	2.51
GA (Binary)	1200.00	0.0020	0.0783	12.36	<b>63.88</b>	30.03	2.25
GA (Binary)	1020.42	0.0020	0.0628	15.18	50.05	<b>51.74</b>	1.54
GA (Binary)	1074.73	0.0035	0.0819	24.16	53.80	30.01	<b>3.70</b>
GA (FP)	1053.00	0.0020	0.1000	<b>11.28</b>	62.93	30.01	2.53
GA (FP)	1053.00	0.0020	0.1000	11.28	<b>62.93</b>	30.01	2.53
GA (FP)	1134.11	0.0020	0.0500	16.66	50.01	<b>53.43</b>	1.36
GA (FP)	952.98	0.0042	0.0960	28.57	50.09	30.42	<b>4.61</b>
Literature	1048.0	0.0020	0.1000	<b>11.30</b>	62.65	30.29	2.51
Literature	1200.0	0.002	0.0776	12.43	<b>63.64</b>	30.31	2.23
Literature	840.0	0.002	0.1000	12.46	51.11	<b>46.52</b>	2.02
Literature	944.0	0.004	0.1000	25.60	51.16	30.38	<b>4.40</b>

Table 6.3: Comparison of results computing the ideal vector of example 2 from Chapter 5 (machining recommendations). For each method the best results for each objective function are shown in **boldface**. OS stands for Osyczka's Multi-objective Optimization System.

Method	$v$ (sfm)	$f$ (ipr)	$d$ (in)	SR ( $\mu$ in)
Ideal Vector				11.28
Monte Carlo 1	914.37	0.0030	0.0969	19.55
Monte Carlo 2	1014.64	0.0026	0.0739	18.77
Min-max (OS)	1200.00	0.0026	0.0499	21.95
GCM (OS)	1200.00	0.0026	0.0499	21.95
WMM (OS)	1200.00	0.0026	0.0499	21.95
PMM (OS)	1200.00	0.0016	0.0496	13.02
NMM (OS)	1200.00	0.0016	0.0496	13.02
GALC (B)	1200.00	0.0044	0.0607	36.03
GALC (FP)	987.57	0.0020	0.1000	11.60
Lexicographic (B)	1200.00	0.0020	0.0657	13.76
Lexicographic (FP)	1200.00	0.0020	0.0759	12.60
VEGA (B)	1200.00	0.0020	0.0781	12.38
VEGA (FP)	1037.39	0.0020	0.1000	11.35
NSGA (B)	1200.00	0.0020	0.0691	13.34
NSGA (FP)	959.16	0.0020	0.1000	11.75
MOGA (B)	1118.34	0.0021	0.0868	12.67
MOGA (FP)	1200.00	0.0020	0.0740	12.79
NPGA (B)	1075.07	0.0021	0.0939	12.29
NPGA (FP)	1094.35	0.0020	0.0927	11.61
Hajela (B)	1045.55	0.0020	0.1000	11.31
Hajela (FP)	1200.00	0.0034	0.0556	28.19
GAminmax1 (B)	1082.18	0.0020	0.0943	11.55
GAminmax1 (FP)	1049.50	0.0020	0.1000	11.29
GAminmax2 (B)	1041.07	0.0020	0.1000	11.33
GAminmax2 (FP)	1029.80	0.0020	0.1000	11.39

Table 6.4: (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the second example (machining recommendations). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective). (Continued in Table 6.5)

Method	SI (% udmg)	TL (min)	MRR (in <sup>3</sup> /min)	$L_p(f)$
Ideal Vector	63.88	53.43	4.61	0.000000
Monte Carlo 1	51.11	36.12	3.18	1.385445
Monte Carlo 2	51.18	40.76	2.35	1.279883
Min-max (OS)	50.49	44.43	1.86	1.889006
GCM (OS)	50.49	44.43	1.86	1.889006
WMM (OS)	50.49	44.43	1.86	1.889006
PMM (OS)	54.18	51.14	1.18	1.856245
NMM (OS)	54.18	51.14	1.18	1.856245
GALC (B)	50.47	30.89	3.84	2.944513
GALC (FP)	59.32	33.98	2.37	0.280571
Lexicographic (B)	59.24	36.05	1.89	0.745638
Lexicographic (FP)	63.03	31.02	2.19	0.295288
VEGA (B)	63.81	30.11	2.25	0.230000
VEGA (FP)	62.07	30.89	2.49	0.064436
NSGA (B)	60.54	34.20	1.99	0.586649
NSGA (FP)	57.75	35.96	2.30	0.411813
MOGA (B)	62.10	30.49	2.45	0.184670
MOGA (FP)	62.35	31.85	2.13	0.371517
NPGA (B)	61.94	30.33	2.54	0.122571
NPGA (FP)	63.11	30.13	2.43	0.073782
Hajela (B)	62.52	30.42	2.51	0.030583
Hajela (FP)	50.65	36.57	2.72	1.990499
GAminmax1 (B)	62.92	30.25	2.45	0.063389
GAminmax1 (FP)	62.73	30.20	2.52	0.014317
GAminmax2 (B)	62.27	30.68	2.50	0.049133
GAminmax2 (FP)	61.65	31.33	2.47	0.096331

Table 6.5: (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the second example (machining recommendations). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective).

was that most methods start with a very small set of feasible solutions (from 3 to 5) and eventually produce complete populations of non-dominated individuals in only a few generations. That, of course, is not the case of in methods, since they start with a complete population of feasible solutions. The behavior of all techniques is more or less the same as before. This time, VEGA produced a very good overall solution, although its final distribution totally converges to that single vector. MOGA and NPGA presented very good performances, considering that they do not manipulate objective function values directly. Hajela's method produced an excellent overall result, but also presented a premature convergence of the entire population to that single vector. It should be noted that the poor performance of floating point representation for Hajela's method is due to a problem with the definition of ranges for the weights. As in the previous example, my weighting min-max method produced the best overall results using floating point representation, and my min-max method that uses sharing presented very good solutions with both binary and floating point representations. It is important to mention that because of the quick convergence property of floating point representation, it was necessary to use a very small sharing factor to avoid total convergence of the population to a single value. A value of  $\sigma_{share} = 0.01$  gave the best distribution, and a value of  $\sigma_{share} = 0.001$  gave the best overall result, presented in Tables 6.4 and 6.5. When binary representation was used, as with all the previous methods, a value of  $\sigma_{share} = 1.0$  was used. It is interesting how even when the best solution produced is very close to the ideal vector, several methods (remarkably several mathematical programming techniques) failed to find it.



Method	$x_1$	$x_2$	$x_3$	$x_4$	$f_1$	$f_2$
Monte Carlo 1	59.08	189.17	90	75	<b>606667.43</b>	0.032467
Monte Carlo 1	26.26	193.29	90	85	1457744.67	<b>0.019242</b>
GA (Binary)	60.00	200.00	80	75	<b>466532.80</b>	0.038087
GA (Binary)	25.00	190.09	95	90	1640191.80	<b>0.016613</b>
GA (FP)	56.16	194.49	95	90	<b>312430.43</b>	0.017951
GA (FP)	25.35	189.58	95	90	1641135.80	<b>0.016615</b>
Literature	63.89	183.29	85	80	<b>531059.8</b>	0.030182
Literature	66.45	183.36	95	85	694101.0	<b>0.023078</b>

Table 6.6: Comparison of results computing the ideal vector of example 3 from Chapter 5 (design of a machine tool spindle). For each method the best results for optimum  $f_1$  and  $f_2$  are shown in **boldface**.

## 6.4 Example 3 : Design of a Machine Tool Spindle

Again, I will start by showing the feasible region in the objective function space for this example, so that we can visualize the Pareto front that we wish to achieve. Figure 6.42 shows the feasible region in the objective function space for this problem. In this case, the Pareto front is very similar to the graph of the first example, since this is also a minimization problem and the best compromises will be at the bottom part of the curve.

The ideal vector of this problem was computed using Monte Carlo Methods 1 and 2 (generating 100 points) presented in Chapter 4, and a GA (with a population of 100 chromosomes running during 50 generations) using binary and floating point representation, with the procedure described in Chapter 4 to adjust its parameters. The corresponding results are shown in Table 6.6, including the best results reported in the literature [192]. The results for Monte Carlo Method 2 are the same than for Method 1. Notice that *Osyczka's multiobjective optimization system* is not able to handle discrete variables, so no results are available for the min-max method using Osyczka's system. The GA using both binary and

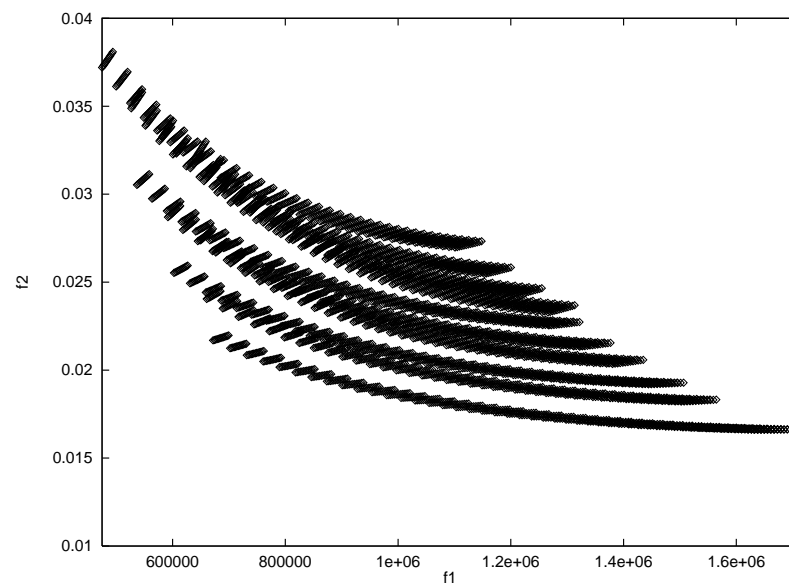


Figure 6.42: Example 3: Initial feasible region for example 3.

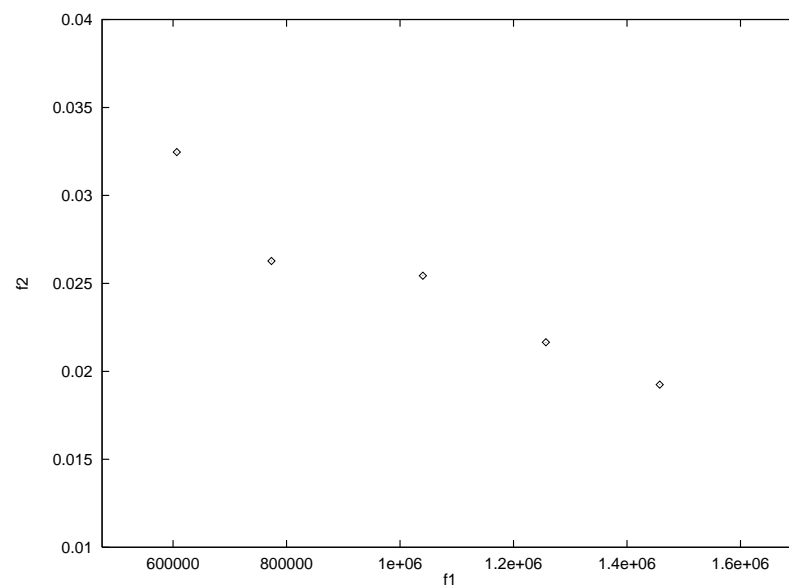


Figure 6.43: Example 3: Initial feasible region for Monte Carlo method solving the third example.

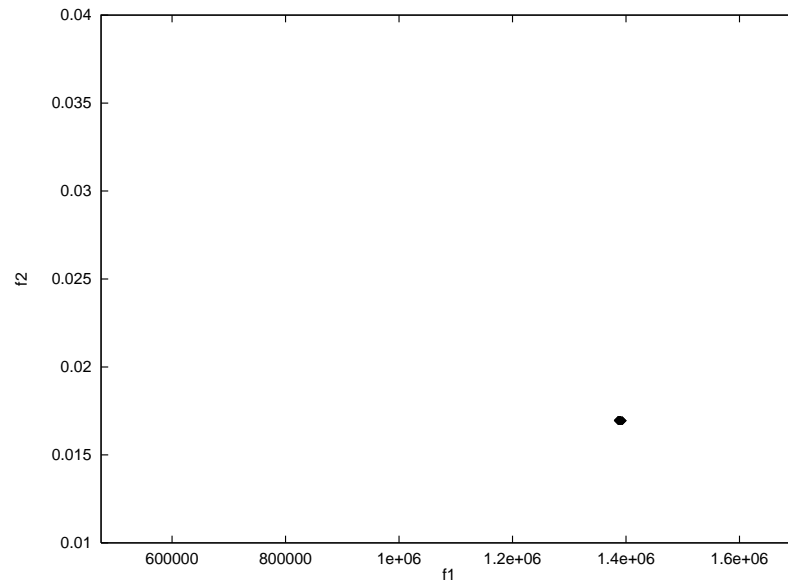


Figure 6.44: Example 3: The GA using a linear combination of the objectives with scaling, after 50 generations using binary representation.

floating point representation found the ideal vector. As can be seen in the results, the best result for the second objective function was found using binary representation, but the solution generated using floating point representation is extremely close, so it is reasonable to assume that the GA using floating point representation found a better solution than the one that used binary representation.

Since the Monte Carlo Methods previously mentioned and Osyczka's multiobjective optimization system do not generate the Pareto front, I will compare them with the GA-based approaches only in terms of the best overall result found. Besides those results, it is interesting to observe the initial distribution of points randomly generated by the method (see Figure 6.43). As should be expected of a highly constrained problem, there are very few feasible points in the initial random set generated (only five), and this will be observed also with the GA-based methods that do not check feasibility. Nevertheless, as in previous examples, the

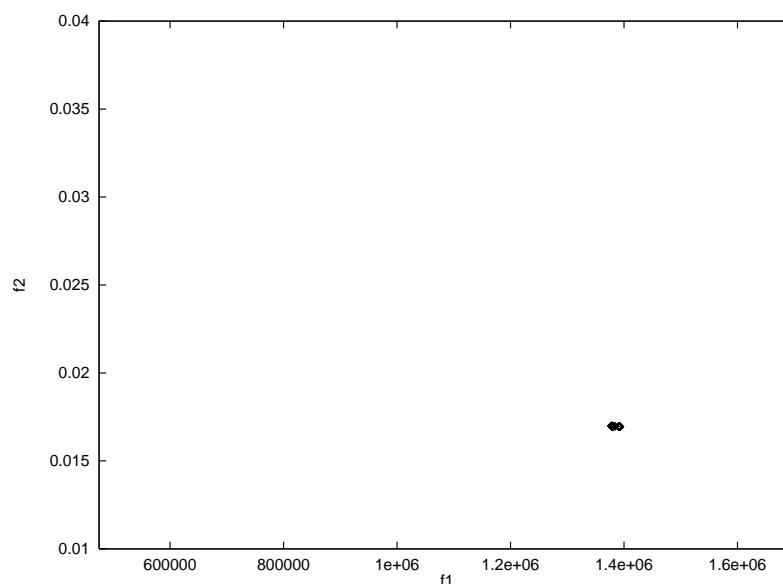


Figure 6.45: Example 3: The GA using a linear combination of the objectives with scaling, after 50 generations using floating point representation.

GA is able to generate the Pareto front in a single run, whereas most mathematical programming techniques require a great deal of effort to do that, when possible, or simply generate a single final solution (the best overall).

We will start by analyzing the behavior of the **Lexicographic Method** at different stages of the search process. Figure 6.46 shows the distribution of points produced at generation zero using this method. There are not many points, but there are certainly more to start than when using Monte Carlo method. Figure 6.47 shows the distribution of points at generation 20 using binary representation, and Figure 6.48 shows the corresponding distribution using floating point representation. Notice how the points start grouping at the lower part of the Pareto front, but there are still several dominated points in the results. It is interesting to see how the floating point representation provides a more uniform distribution than binary representation, even when using the same method and the same parameters for the GA. Figure 6.48 shows how there are fewer dominated points at generation 20, and the Pareto front seems to be more clear. However,

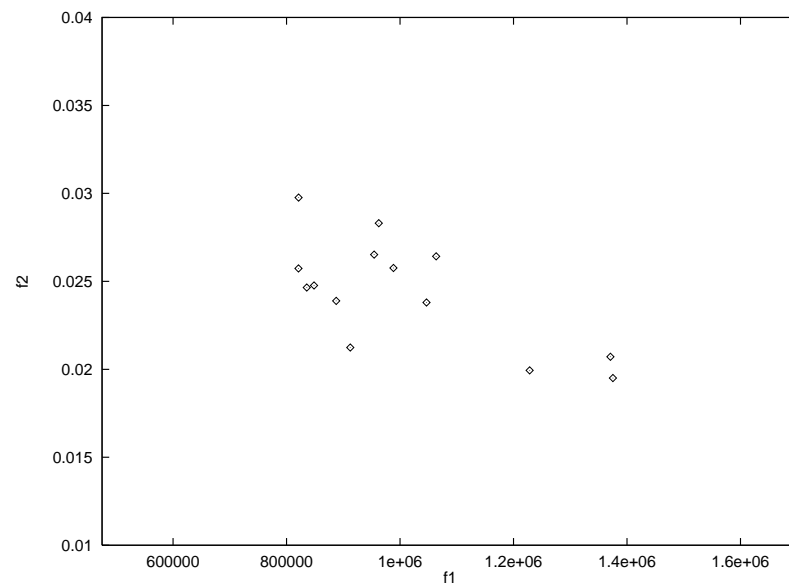


Figure 6.46: Example 3: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed.

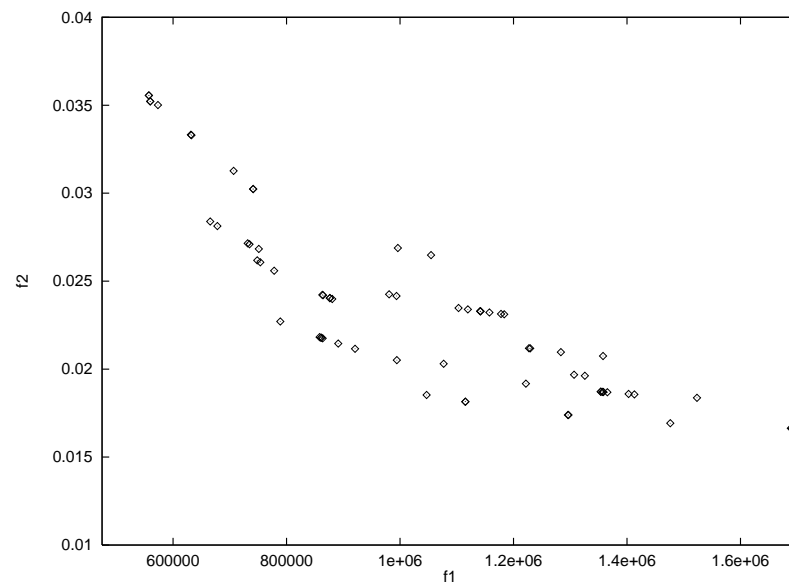


Figure 6.47: Example 3: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed.

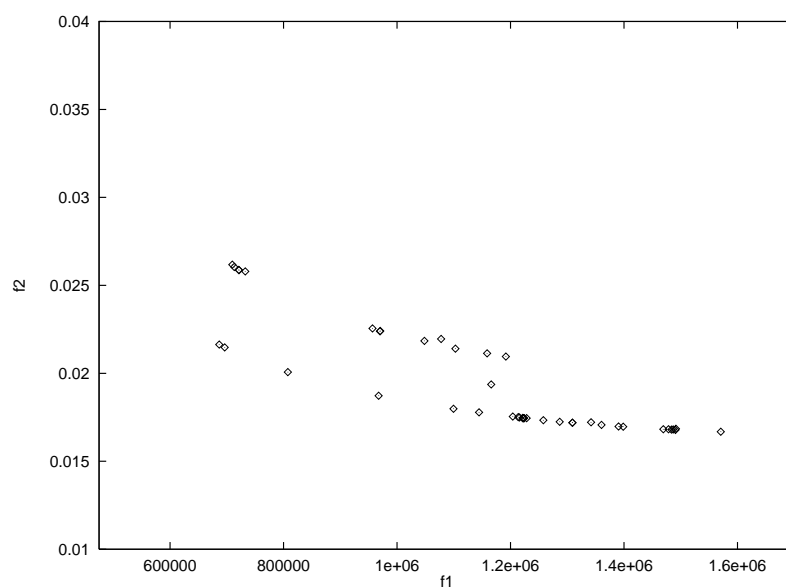


Figure 6.48: Example 3: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed.

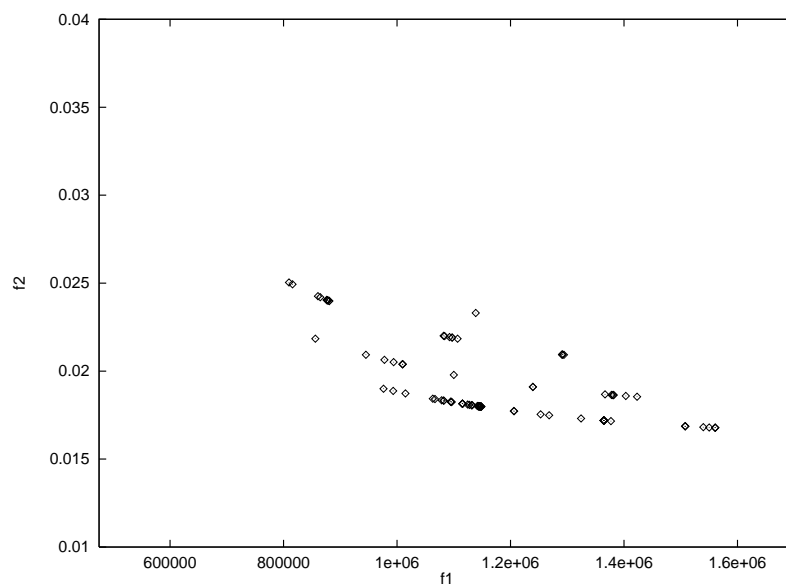


Figure 6.49: Example 3: Distribution of points using the Lexicographic Method with a binary representation at generation fifty. Only points within the feasible region are displayed.

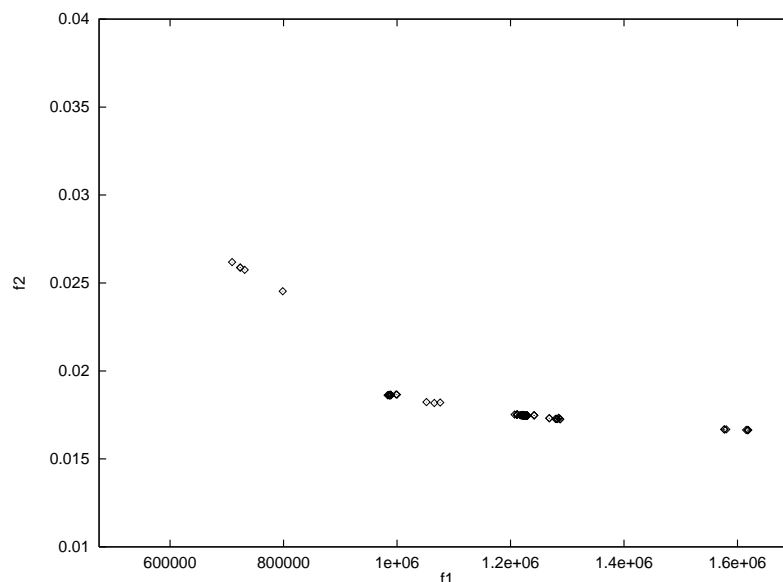


Figure 6.50: Example 3: Distribution of points using the Lexicographic Method with a floating point representation at generation fifty. Only points within the feasible region are displayed.

after 50 generations, binary representation seems to produce a better distribution (see Figure 6.49) than floating point representation (see Figure 6.50). There are also fewer points displayed, because there is more redundancy of solutions using floating point representation. The reason for this behavior is that this representation scheme provides a faster convergence, and therefore the population at this stage of the search is starting to converge towards a global optimum. Notice, however, that when the GA uses floating point representation a better overall solution is found. I also compared the effect of a simple linear combination of objectives (addition or multiplication) using scaling, and the results after 50 generations are shown in Figure 6.44. Observe how in this case, even a good scaling of the objectives is not able to find a good trade-off, and, as expected, is also unable to delineate the Pareto front. The use of a floating point representation slightly improves the best overall solution, although we still get global convergence of the

Method	$x_1$	$x_2$	$x_3$	$x_4$	$f_1$	$f_2$	$L_p(f)$
Ideal Vector					312430.43	0.016613	0.000000
Monte Carlo 1	56.67	190.22	85	80	728581.78	0.026474	1.925552
Monte Carlo 2	26.26	193.29	90	85	1457744.67	0.019242	3.824071
GALC (B)	42.27	187.83	95	90	1386131.13	0.016955	3.457194
GALC (FP)	42.78	188.01	95	90	1377893.38	0.016975	3.432031
Lexicographic (B)	62.02	200.00	95	85	856072.60	0.021843	2.054856
Lexicographic (FP)	61.98	190.91	95	80	709307.00	0.026191	1.846824
VEGA (B)	54.63	200.00	90	85	987526.38	0.021241	2.439365
VEGA (FP)	54.45	191.11	95	90	1151553.50	0.017747	2.754052
NSGA (B)	65.22	200.00	90	85	708412.19	0.024386	1.735100
NSGA (FP)	62.00	197.36	95	90	985238.13	0.018839	2.287456
MOGA (B)	65.52	200.00	90	85	699786.88	0.024531	1.716431
MOGA (FP)	67.75	189.34	95	90	800608.63	0.020108	1.772895
NPGA (B)	57.92	200.00	90	75	654768.06	0.032233	2.035952
NPGA (FP)	43.53	187.86	95	90	1363536.50	0.017006	3.387944
Hajela (B)	59.87	188.12	95	80	757841.81	0.024983	1.929456
Hajela (FP)	61.19	188.10	95	90	975296.19	0.018607	2.241669
GAminmax1 (B)	66.99	200.00	90	85	656950.38	0.025319	1.626757
GAminmax1 (FP)	71.98	188.17	95	90	672894.56	0.021687	1.459166
GAminmax2 (B)	58.58	192.94	95	85	926272.00	0.020756	2.214113
GAminmax2 (FP)	69.47	191.78	95	90	759919.19	0.020682	1.677212

Table 6.7: Comparison of the best overall solution found by each one of the methods included in MOSES for the third example (design of a machine tool spindle). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.5 (equal weight for every objective).

population (see Figure 6.45). This example shows how sometimes the simple linear combination of objectives using scaling completely fails to find a reasonable good solution to a multiobjective optimization problem.

Schaffer's **VEGA** produces a rough approximation of a part of the Pareto front after 20 generations using binary representation (see Figure 6.51), but it still has several dominated points within the population. After 50 generations, however, the contour seems very well defined, and we can clearly see the two lower



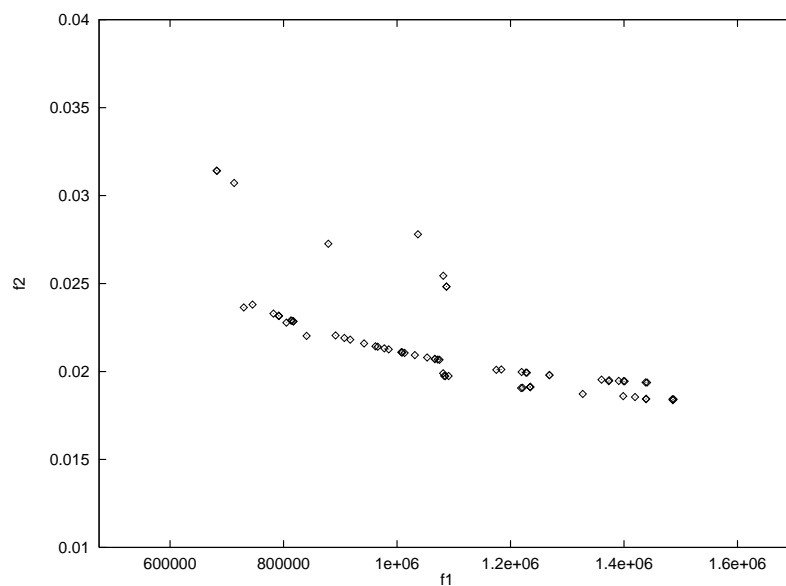


Figure 6.51: Example 3: Distribution of points using VEGA with a binary representation at generation twenty.

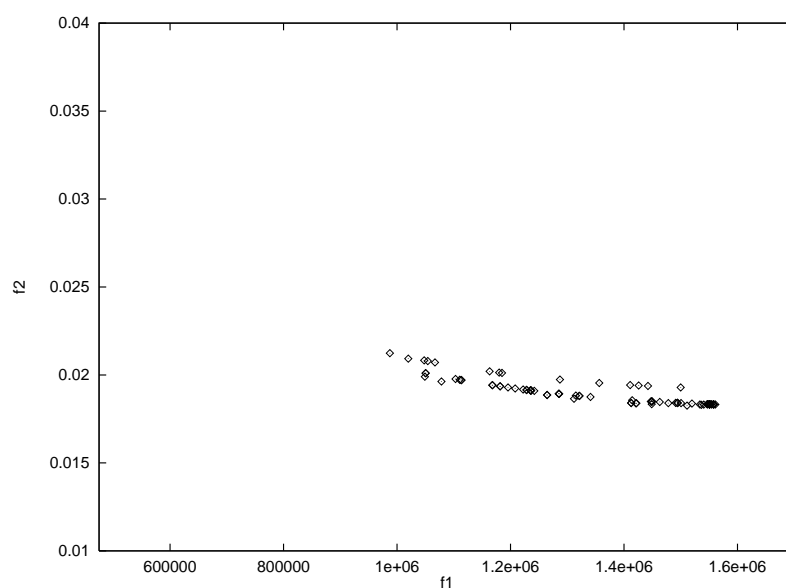


Figure 6.52: Example 3: Distribution of points using VEGA with a binary representation at generation fifty.

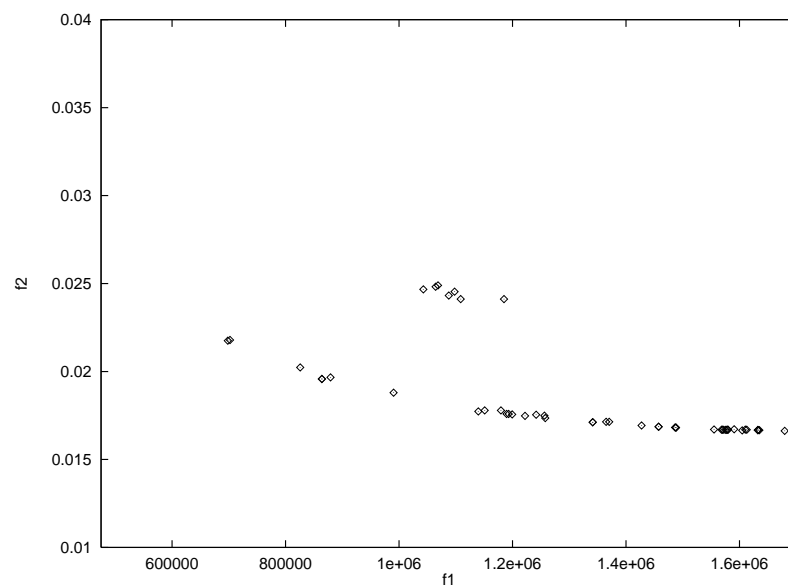


Figure 6.53: Example 3: Distribution of points using VEGA with floating point representation at generation twenty.

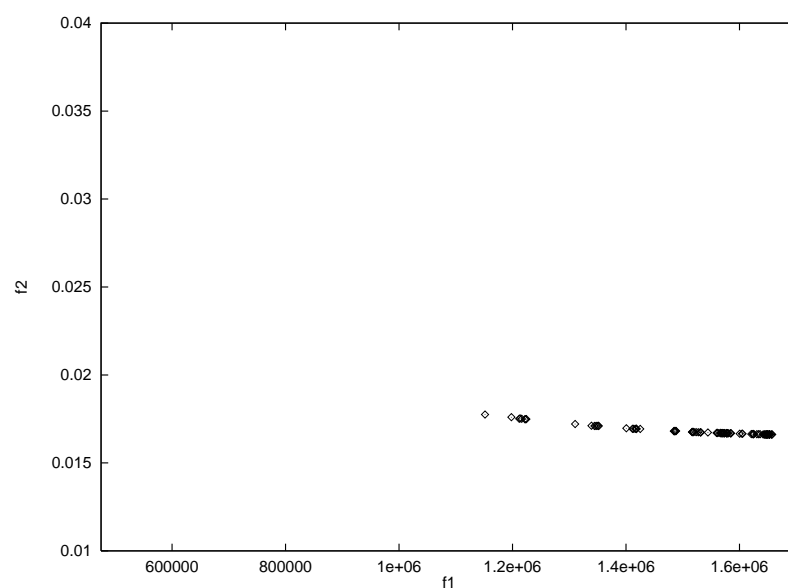


Figure 6.54: Example 3: Distribution of points using VEGA with floating point representation at generation fifty.

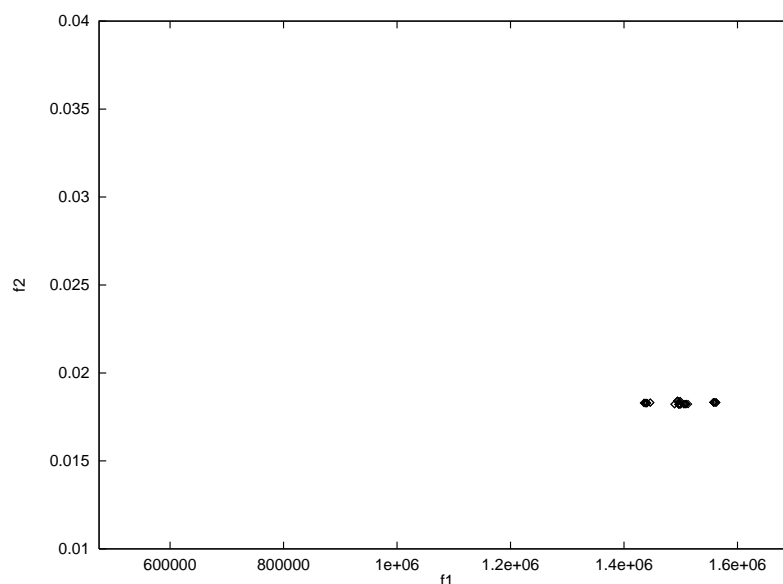


Figure 6.55: Example 3: Distribution of points using VEGA with binary representation at generation 100.

curves of the feasible zone (see Figure 6.52). Even when a good portion of the Pareto front can be generated using VEGA, the best overall solution produced is not very good, since the method did not keep points near the corner of the inferior curve where the best trade-off is located. Floating point representation has a very promising start (see Figure 6.53), and it converges to a reasonably good Pareto contour that is, however, not as good as that produced with binary representation. Again, the quick convergence property of this scheme representation produces a much more stable contour, but with fewer points (see Figure 6.54). Interestingly, in this example we have again the case of an overall solution poorer than that produced using binary representation. The reason is the same mentioned in the first example: the population started converging towards a Pareto solution at the bottom right of the Pareto front, instead of converging towards the left angle of the contour. Finally, as in previous cases, VEGA converges to very few non-dominated solutions after only 100 generations (see Figure 6.55), again indicating

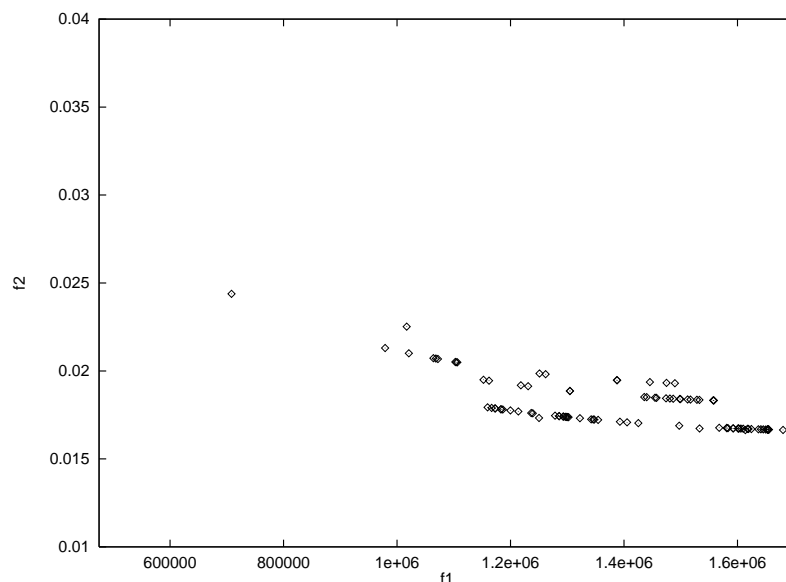


Figure 6.56: Example 3: Distribution of points using NSGA with binary representation at generation fifty.

this method's tendency toward global convergence of the population to a single point.

My version of Srinivas' **NSGA** produces a rough approximation of the Pareto front using binary representation after 50 generations, as can be seen in Figure 6.56. The three inferior curves are quite clear. However, it seems that the technique still requires more generations to converge towards a better contour. If we monitor its behavior after only 20 generations, we will see a very sparse distribution of points (see Figure 6.57). However, if the number of generations is excessive, there is a trend towards global convergence, as can be observed in Figure 6.58. This effect can be again anticipated by using floating point representation with only 50 generations (see Figure 6.59). This curve is about the best contour that we can get using this technique in this example. It is interesting that we were able to produce a very good trade-off solution using binary representation, but this is not a trend when using this technique, and, as mentioned before, the GA

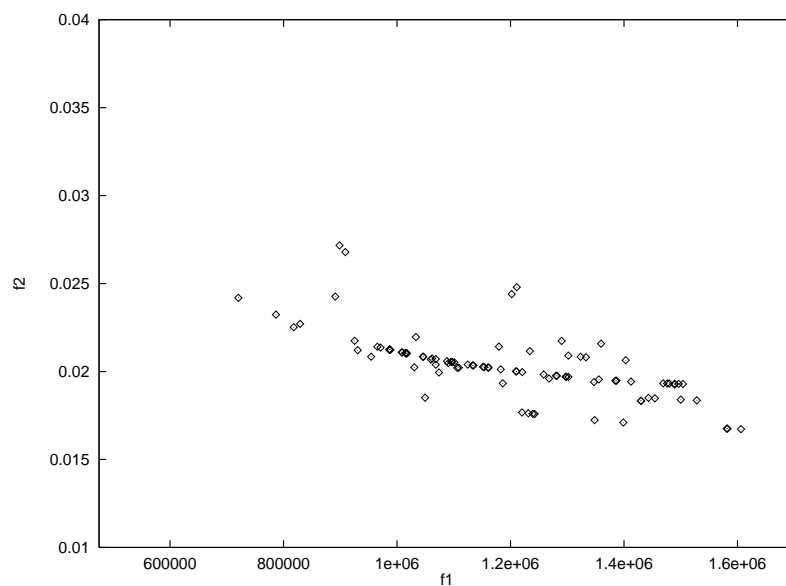


Figure 6.57: Example 3: Distribution of points using NSGA with binary representation at generation twenty.

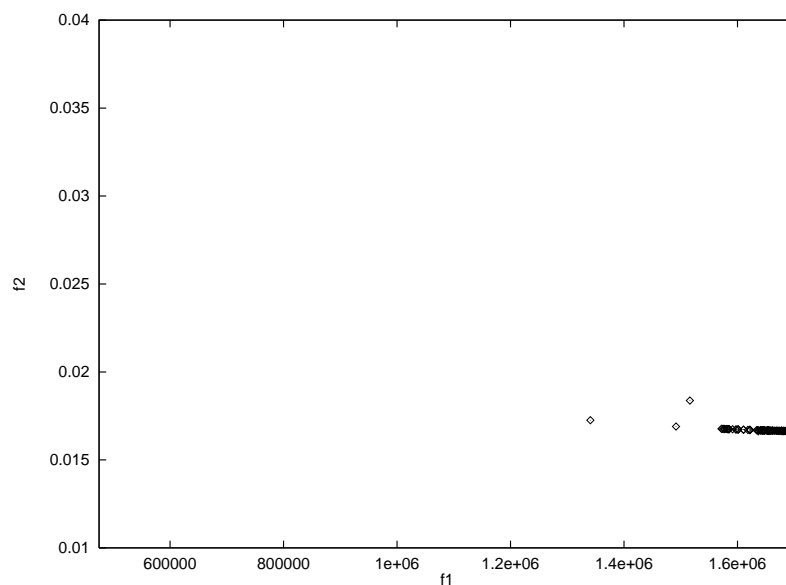


Figure 6.58: Example 3: Distribution of points using NSGA with binary representation at generation 500.

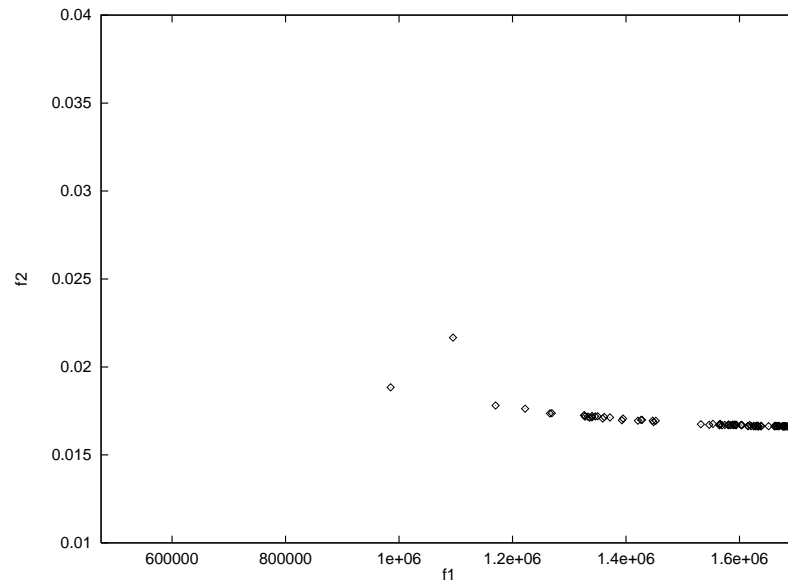


Figure 6.59: Example 3: Distribution of points using NSGA with floating point representation at generation 50.

is not able to keep it over generations. Finally, there was not significant change in the results when the value of  $\sigma_{share}$  was modified.

Fonseca's MOGA shows once again a very interesting behavior. First, running the GA with a binary representation during 20 generations, we can see an sparse but clear distribution of points (see Figure 6.60) that start delineating the Pareto contour. As in the first example, after 50 generations the contour is excellent, but there are still several dominated points in the graph (see Figure 6.61). After 100 generations, we start losing the contour, and there is a strong trend to converge to a single point (see Figure 6.62). After 500 generations, the entire population converged to only 3 almost identical non-dominated points, as in the first example. In this case, the best overall solution is better than all the other previous techniques for both representations. It is also interesting to notice that in this example, a floating point representation produced a poorer overall solution after 50 generations, and a fuzzier Pareto front (see Figure 6.63). I also used a

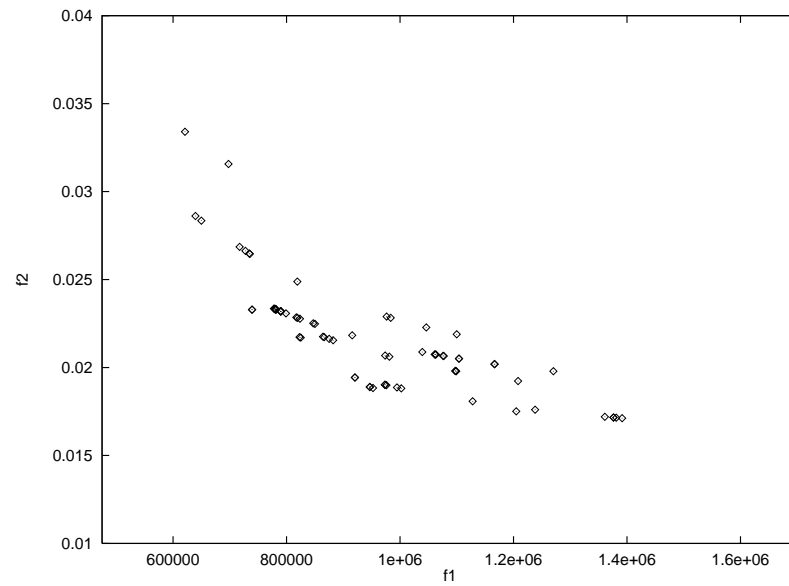


Figure 6.60: Example 3: Distribution of points using MOGA with binary representation at generation 20.

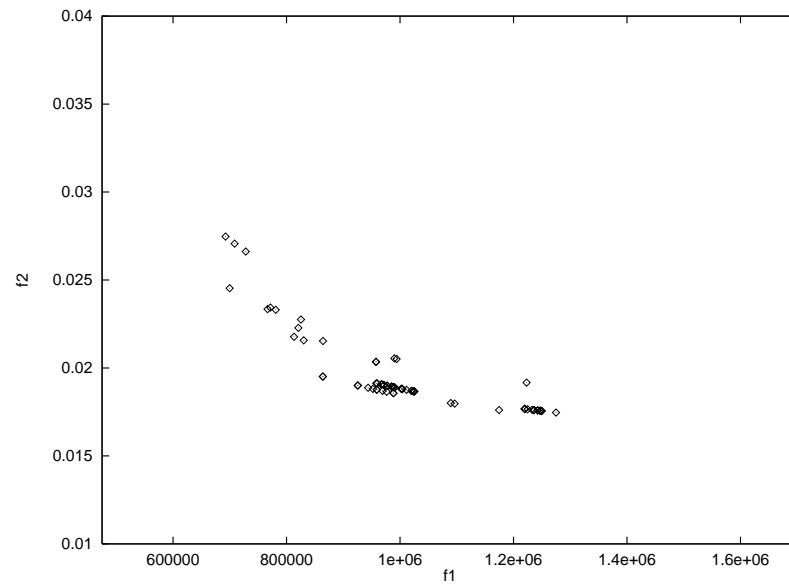


Figure 6.61: Example 3: Distribution of points using MOGA with binary representation at generation 50.

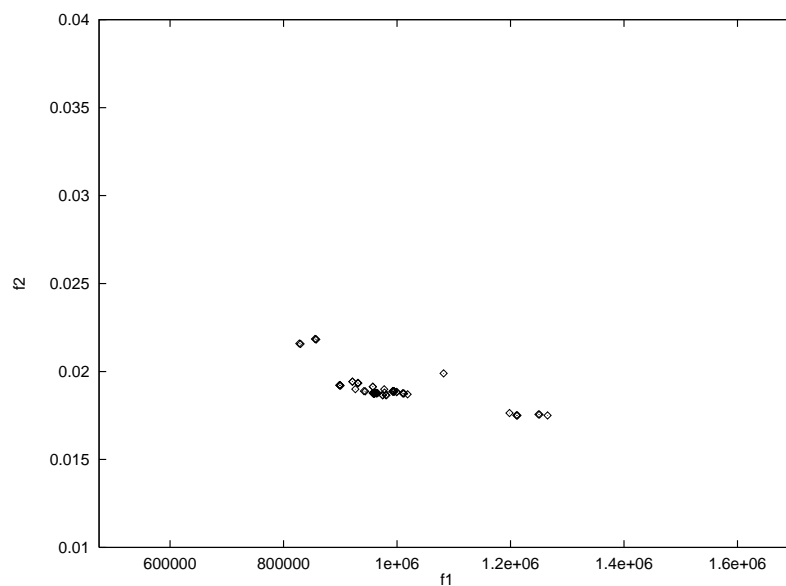


Figure 6.62: Example 3: Distribution of points using MOGA with binary representation at generation 100.

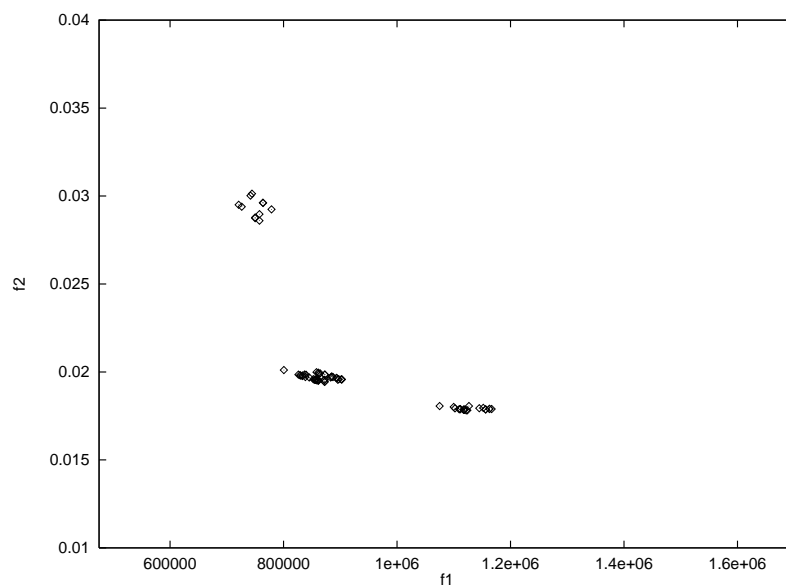


Figure 6.63: Example 3: Distribution of points using MOGA with floating point representation at generation 50.



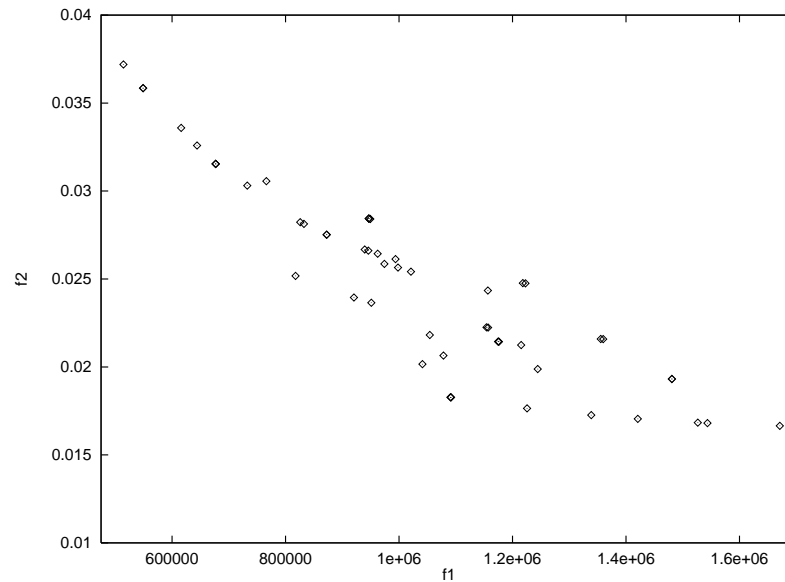


Figure 6.64: Example 3: Distribution of points using NPGA with binary representation at generation 20.

value of  $\sigma_{share} = 0.1$  as in NSGA, and changing this value did not affect the results produced by the algorithm.

Again, NPGA has a very good start, as can be observed in the graph produced after 20 generations (see Figure 6.64), but it still has a lot of dominated points in that search stage. After 50 generations, the distribution is still too sparse, with many dominated individuals (see Figure 6.65). However, if we continue up to 100 generations, the front is more stable and there is no convergence to a single point (see Figure 6.66). Remarkably, after 500 generations, the population looks very well distributed, proving the stability of the algorithm (see Figure 6.67), although the Pareto front does not seem too clear. Once more, this technique is able to keep more points through a lot of generations than any other Pareto-based technique, and its best overall solution using binary representation is remarkably good considering the scarce manipulation of the fitness function. It should be mentioned again that the value of  $\sigma_{share}$  plays a critical role in the performance of the algorithm, because if we increase it from 0.1 (same value used

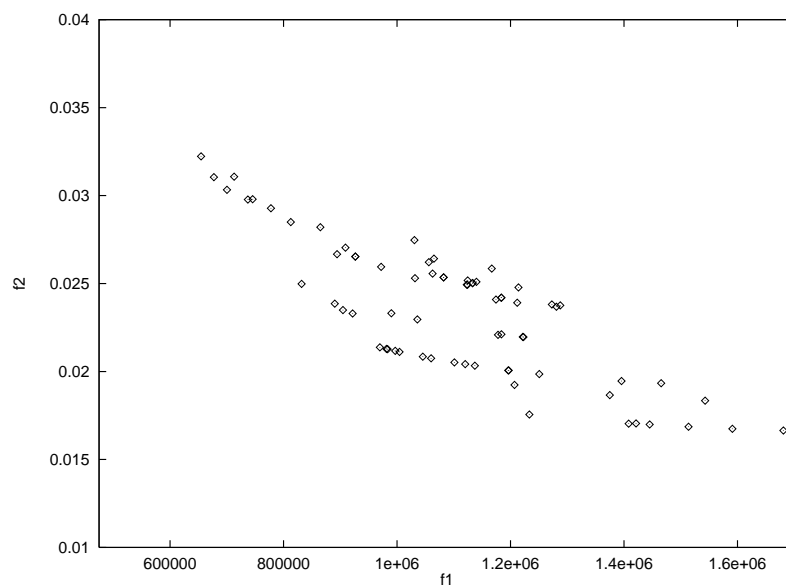


Figure 6.65: Example 3: Distribution of points using NPGA with binary representation at generation 50.

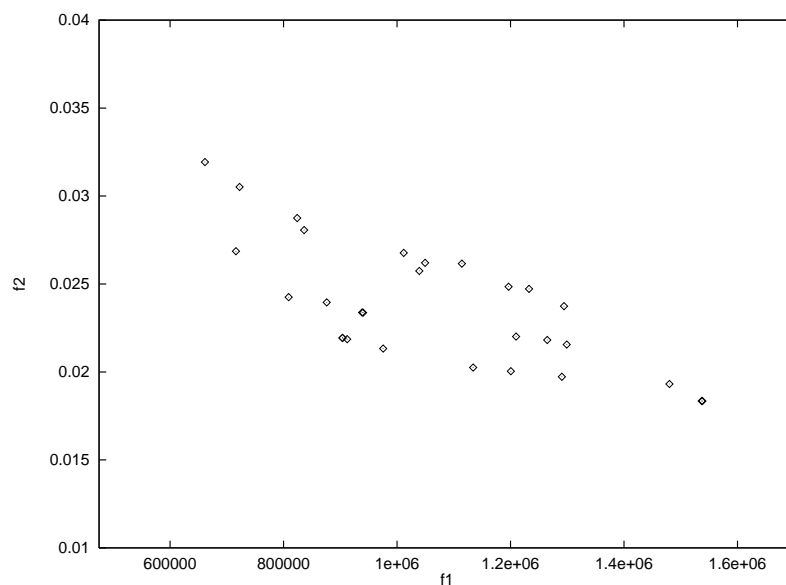


Figure 6.66: Example 3: Distribution of points using NPGA with binary representation at generation 100.

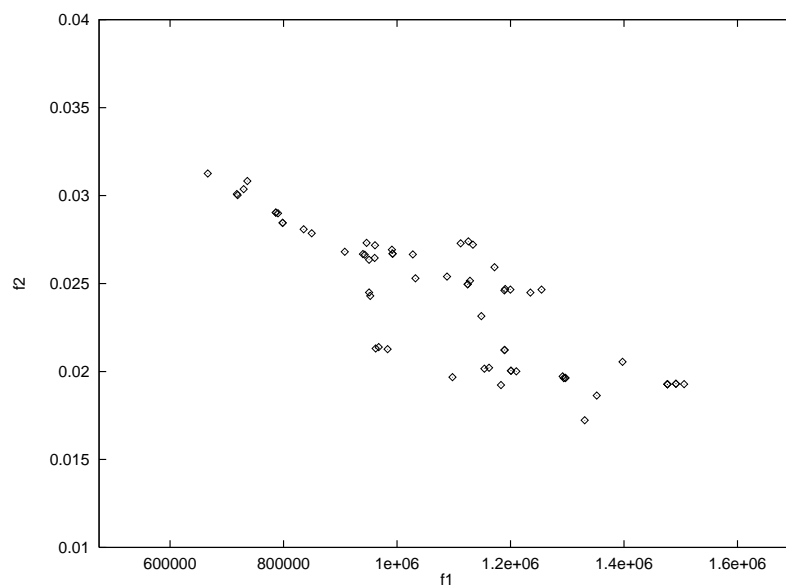


Figure 6.67: Example 3: Distribution of points using NPGA with binary representation at generation 500.

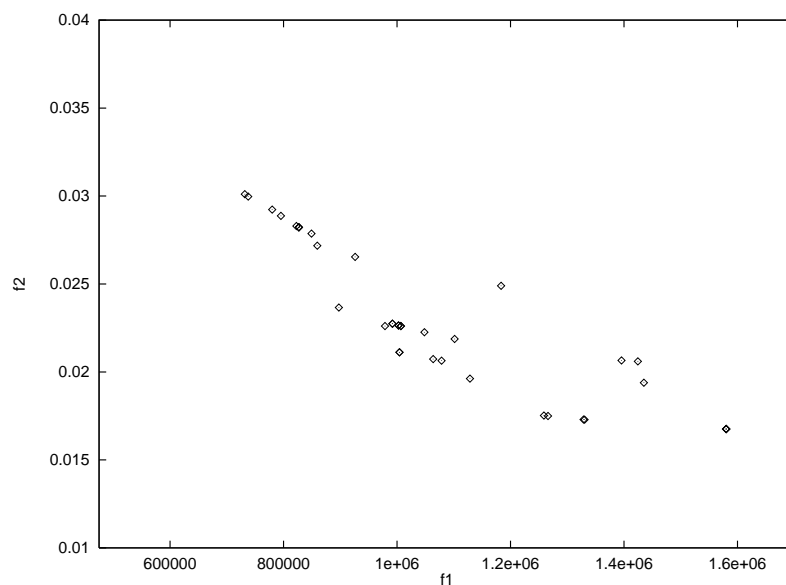


Figure 6.68: Example 3: Distribution of points using NPGA with binary representation at generation 50 with  $\sigma_{share} = 1.0$ .

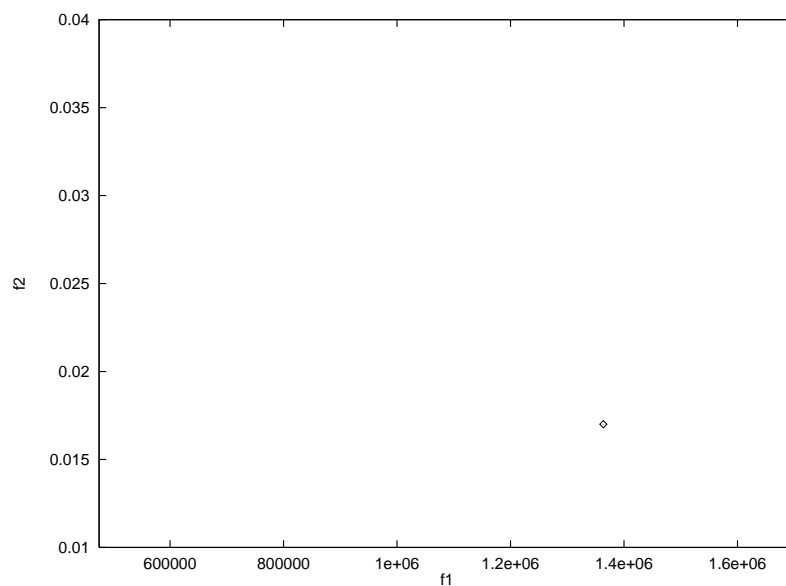


Figure 6.69: Example 3: Distribution of points using NPGA with floating point representation at generation 50.

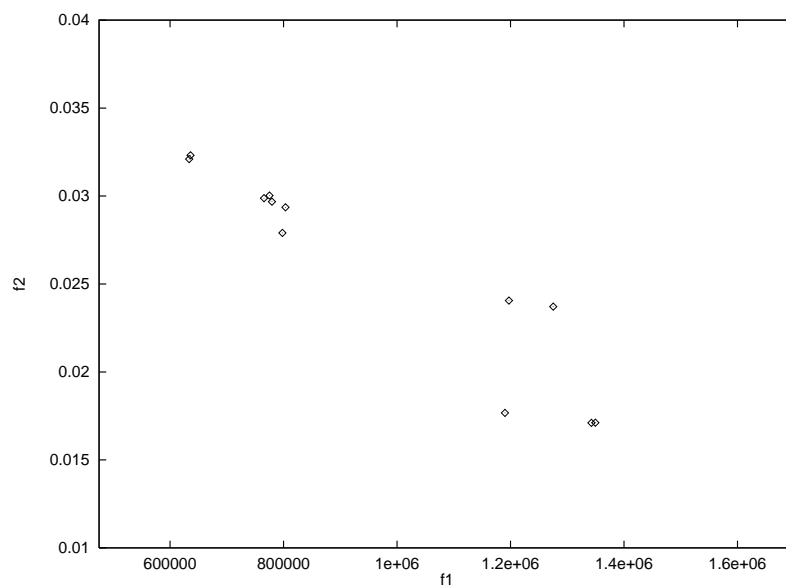


Figure 6.70: Example 3: Distribution of points using NPGA with floating point representation at generation 20.

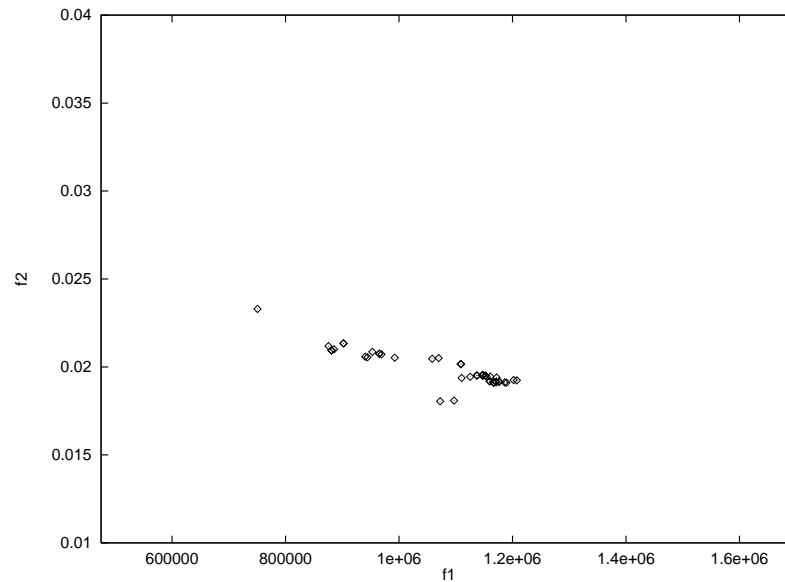


Figure 6.71: Example 3: Distribution of points using Hajela's method with binary representation at generation 20.

in the previous techniques) to 1.0, we will see that the technique has more difficulties to form niches (see Figure 6.68). The value of  $t_{dom}$  significantly affects the performance of the algorithm both in terms of efficiency (it gets much slower) and quality of results (it could completely fail in finding the Pareto front). Another important observation is that a floating point representation did not work at all in this example. The main problem was that the population converged to only infeasible solutions that are good trade-offs but violate the constraints imposed by the problem. Even at early stages of the search (see Figure 6.70) very few feasible points are maintained, and after 50 generations there is only one point left (see Figure 6.69) which is not even a good trade-off. After 100 generations, no feasible solutions appear. This problem is a good example to illustrate what I said before about checking feasibility within these algorithms. If we check at every moment that we are only manipulating feasible solutions, then we can guarantee that our algorithm will keep a population of feasible points and we will not run into the problem presented in this example.

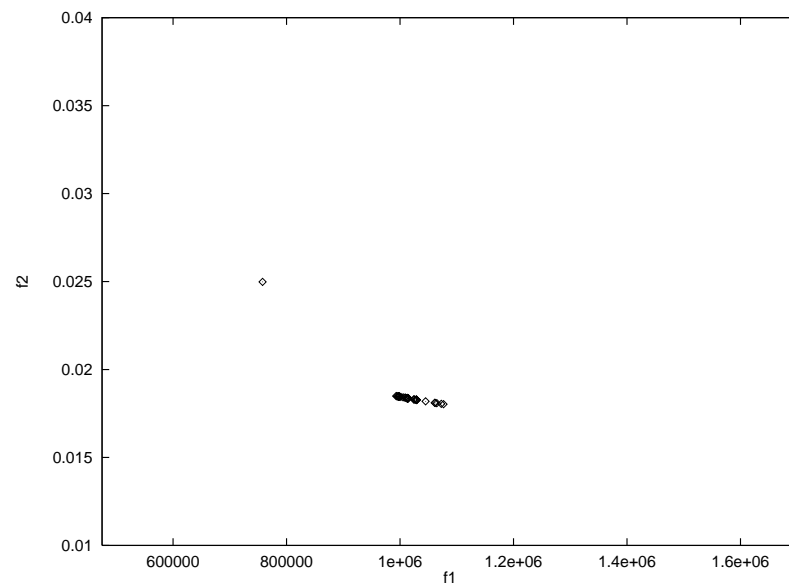


Figure 6.72: Example 3: Distribution of points using Hajela's method with binary representation at generation 50.

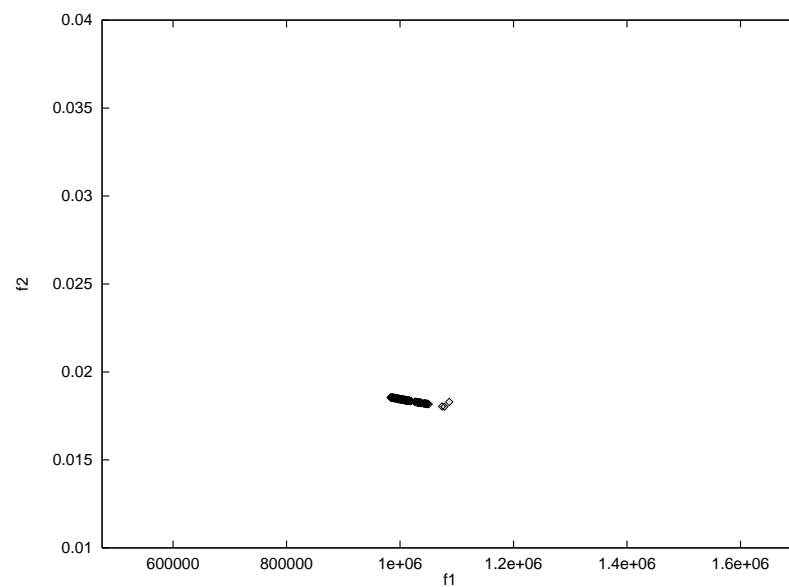


Figure 6.73: Example 3: Distribution of points using Hajela's method with binary representation at generation 50 using 500 individuals.

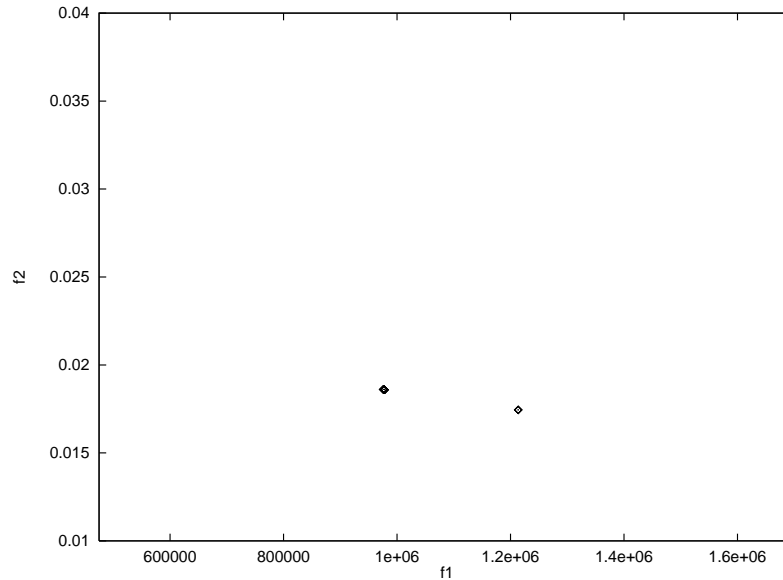


Figure 6.74: Example 3: Distribution of points using Hajela's method with floating point representation at generation 50.

As could be expected from a Non-Pareto approach, the performance of Hajela's method was not very stable with this example. Even after 20 generations, the Pareto front does not seem very clear (see Figure 6.71). Anyway, the performance of the algorithm does not seem too bad until we advance more in the search process. After 50 generations, only a few points are left in the population (see Figure 6.72) producing a good overall solution, but a poor Pareto front. As in the first example, increasing the population size to 500 chromosomes does not affect the result too much (see Figure 6.73). As in the first example, changing the values of  $\sigma_{share}$  and the mating threshold (taken as 0.1 in this case) did not affect the solution in any significant way. In terms of representation schemes, the use of a floating point representation only makes things worse, because the entire population converges, after only 50 generations to a couple of solutions, which turn out to be worse than the best trade-off produced when using a binary representation scheme (see Figure 6.74).

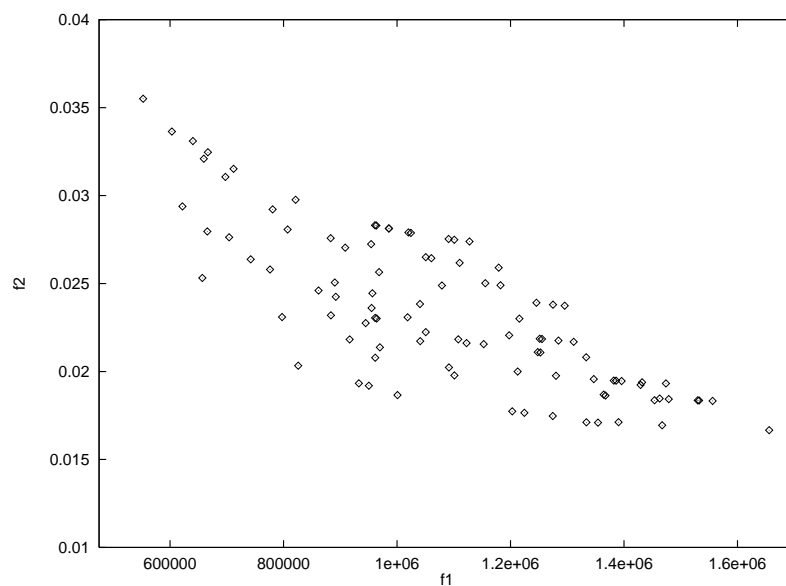


Figure 6.75: Example 3: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero.

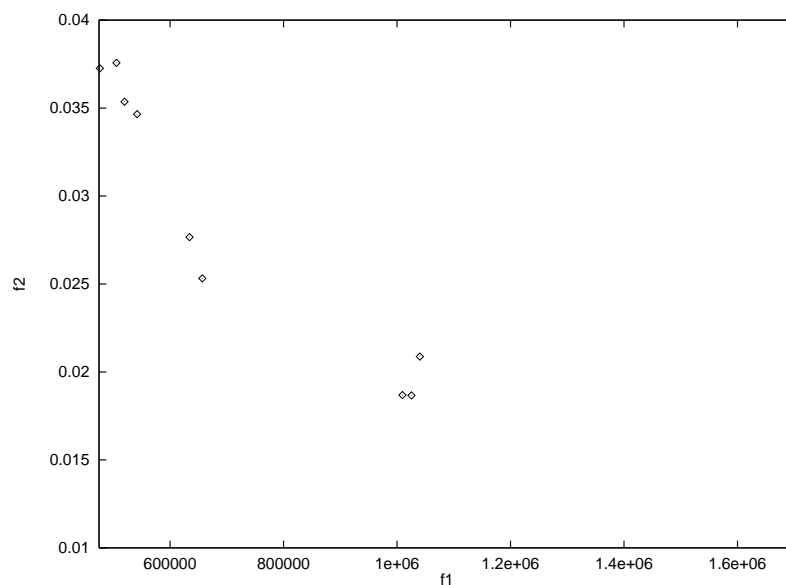


Figure 6.76: Example 3: Distribution of points using my method based on the min-max algorithm with binary representation at generation 50.



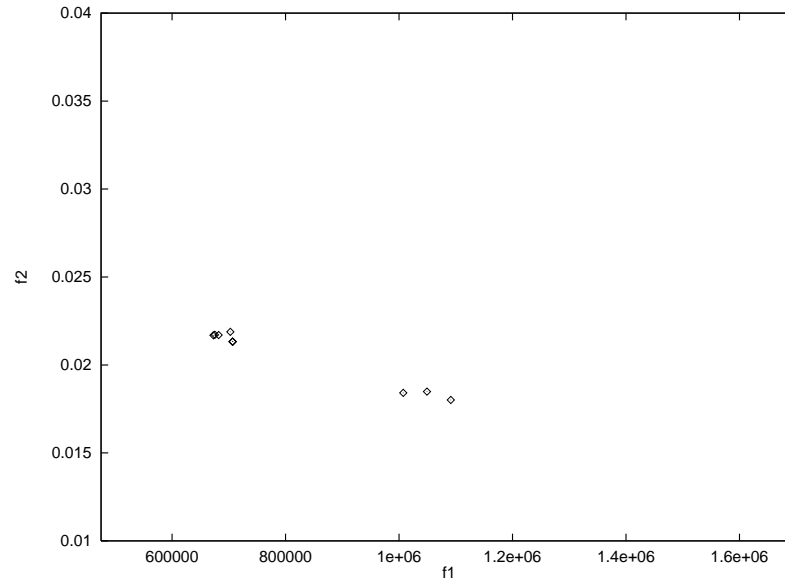


Figure 6.77: Example 3: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 50.

Before showing the results produced with my method based on the min-max approach, I want to show the distribution of points at generation zero, since in this case the algorithm ensures that only feasible solutions are generated. It can be seen in Figure 6.75 that the initial distribution has a fairly large amount of points near the Pareto front, but most of them are distributed over the entire feasible region. After 50 generations, and using 10 different weights (as in the first example), we can get a rough contour of the Pareto front (see Figure 6.76). This example shows the importance of choosing the right weights, since the values that worked very well in the first example do not seem to be very appropriate for this one. However, the best overall solution is the best obtained so far using binary representation (see Table 6.7), and if we connect the dots, we can get a very good approximation of the entire Pareto front, although the use of a different and larger set of weights seems to be more appropriate. The use of floating point representation reduces the number of points in the contour (see Figure 6.77) as in the first example, but it also produces the best trade-off found for this problem.

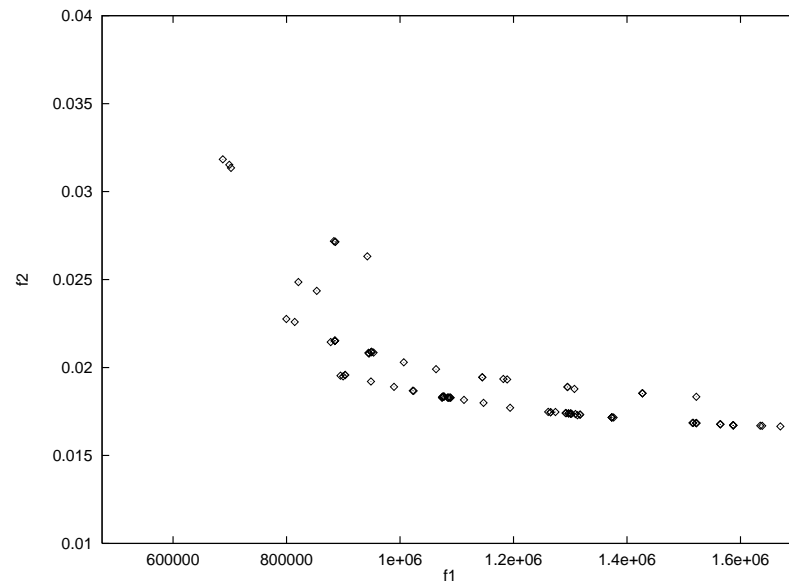


Figure 6.78: Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20.

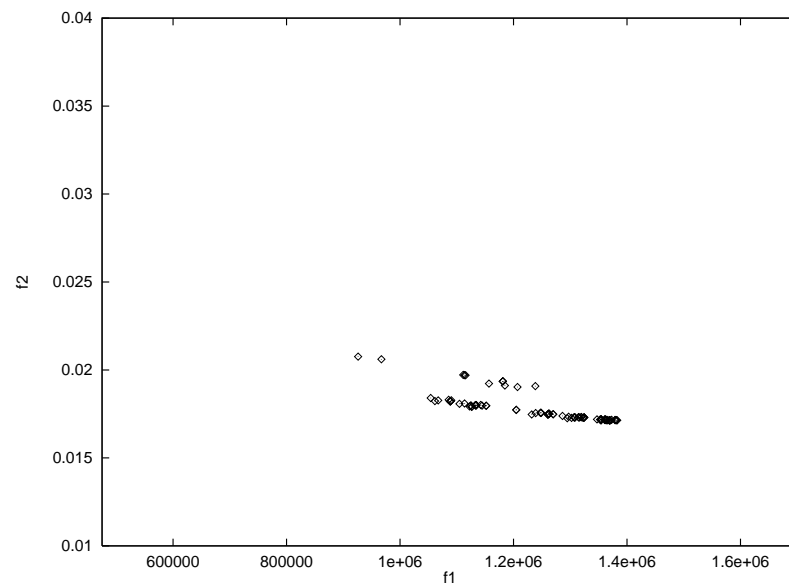


Figure 6.79: Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 50.

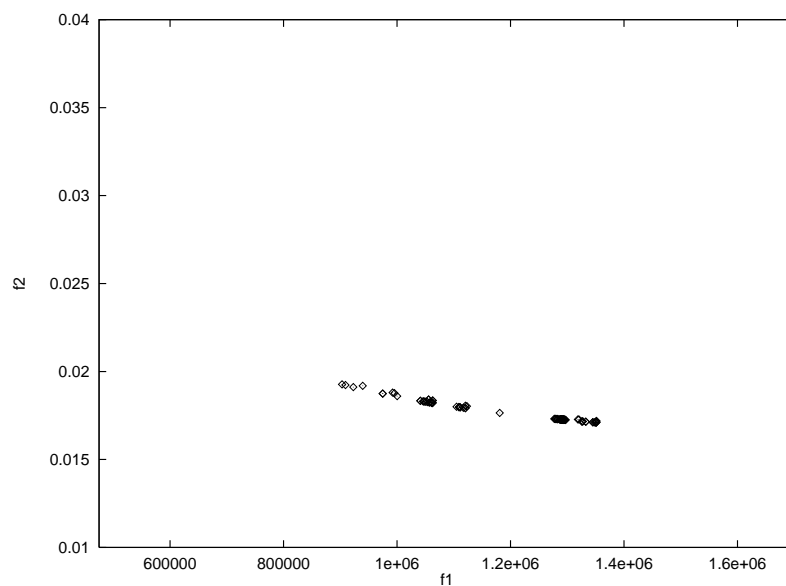


Figure 6.80: Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100.

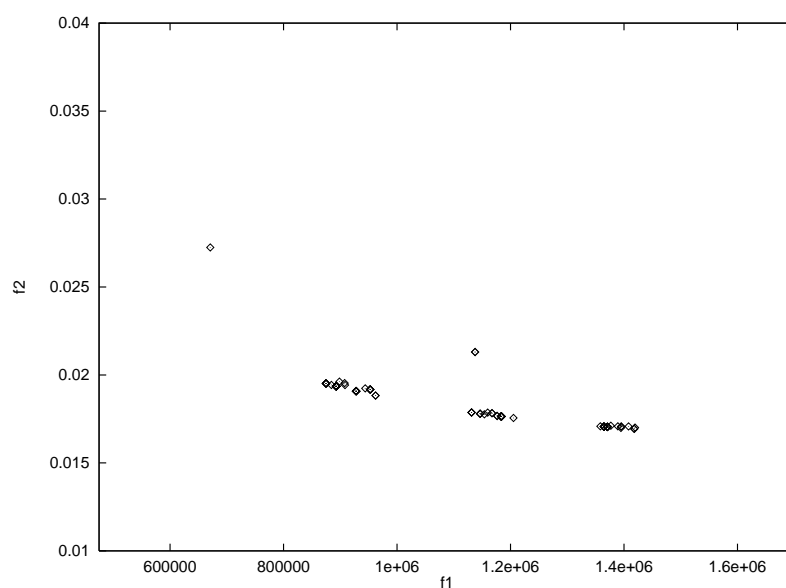


Figure 6.81: Example 3: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 500.

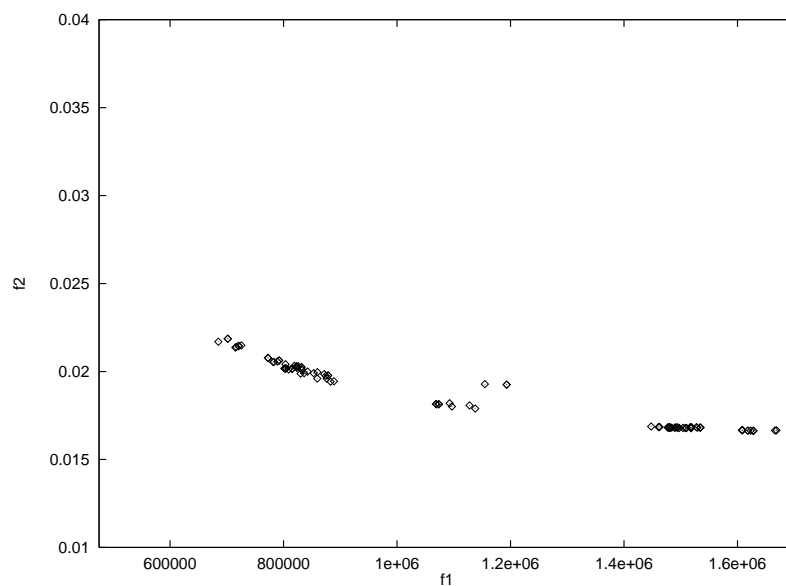


Figure 6.82: Example 3: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20.

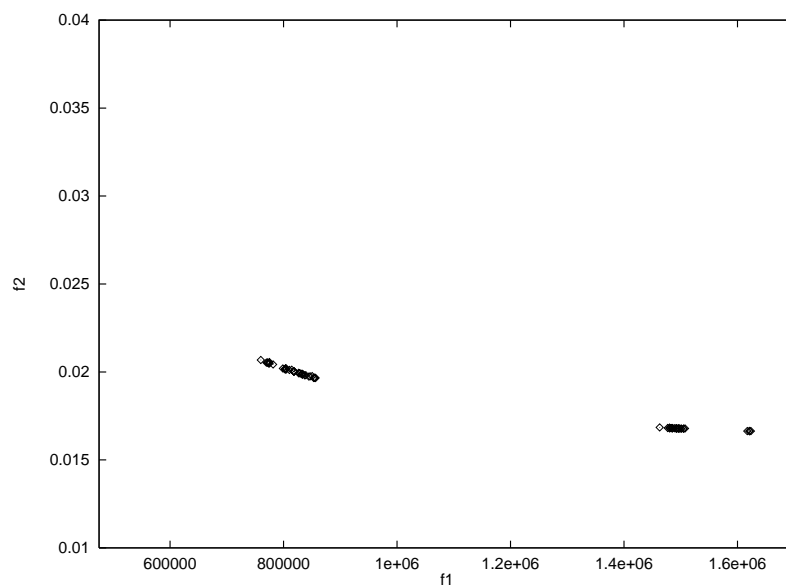


Figure 6.83: Example 3: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 50.

Finally I tried my min-max selection strategy with sharing. Using the same initial population shown in Figure 6.75, this algorithm has a very stable behavior if a good value for  $\sigma_{share}$  is used (I chose 0.5 for this example). After only 20 generations using binary representation, the algorithm starts producing a fairly large portion of the Pareto front, but there are still several dominated points present (see Figure 6.78). At generation 50, the middle part of the Pareto front is very well defined (see Figure 6.79), with relatively few dominated points present in the graph. At Generation 100, most points are grouping within the inferior part of the Pareto curve (see Figure 6.80), forming a smooth line, but without converging to a unique solution. After 500 generations, a few points seem to be spreading again, but a good portion of the Pareto front is still visible (see Figure 6.81), and the population has not converged to a single non-dominated point. The use of a floating point representation produces a very good contour after 20 generations (see Figure 6.82), but tends to converge towards the extremes of the Pareto curve, without being able to generate the middle part of it (see Figure 6.83). With respect to the best overall solution, we have the opposite case of the first example, because floating point representation produces a better result than binary representation. This is because this representation was able to keep a few points near to the corner of global non-dominated individuals. Nevertheless, the graphs show the trend of the method to behave better when binary representation is used.

## 6.5 Example 4 : Design of a 10-bar Plane Truss

Due to the extremely large size of the search space, I will not be able to show it graphically. This is because of the high number of decision variables (20 in total, considering that there are 10 cross-sectional areas to be determined, each consisting of 2 values). Since the range of each variable goes from 0.25 to 999.99,

Method	$x_1$	$x_2$	$x_3$	$f_1$	$f_2$	$f_3$
Monte Carlo 1	32.80	64.55	952.57	<b>31653.40</b>	2.904226	19.813087
Monte Carlo 1	560.66	983.17	476.91	80905.76	<b>0.593976</b>	4.894527
Monte Carlo 1	874.86	974.69	174.33	80641.81	0.593976	<b>4.894527</b>
Min-Max (OS)	149.69	149.69	149.69	<b>18836.86</b>	2.953601	22.275247
Min-Max (OS)	149.94	149.94	149.94	18878.93	<b>2.942967</b>	22.206487
Min-Max (OS)	149.94	149.94	149.94	18878.93	2.942967	<b>22.206487</b>
GA (Binary)	126.84	62.01	0.29	<b>4929.28</b>	7.111559	74.140541
GA (Binary)	999.99	999.99	999.99	115525.01	<b>0.441966</b>	3.503093
GA (Binary)	999.99	974.16	999.99	125884.37	0.441966	<b>3.333183</b>
GA (FP)	98.90	87.60	0.26	<b>4638.28</b>	7.599996	84.149033
GA (FP)	999.99	999.99	999.96	105498.98	<b>0.441966</b>	4.355180
GA (FP)	999.97	999.99	999.95	125882.29	0.441967	<b>3.333209</b>
Literature	68.19	106.40	7.50	<b>4793.20</b>	7.428803	78.343449
Literature	68.19	106.40	7.50	4793.20	<b>7.428803</b>	78.343449
Literature	68.19	106.40	7.50	4793.20	7.428803	<b>78.343449</b>

Table 6.8: (Part I) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). For each method the best results for optimum  $f_1$ ,  $f_2$  and  $f_3$  are shown in **boldface**. OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.9, 6.10 and 6.11)

Method	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$
Monte Carlo 1	218.32	766.84	329.31	309.84	67.62	884.46	238.92
Monte Carlo 1	687.97	651.63	907.20	400.74	527.49	428.28	943.04
Monte Carlo 1	687.97	651.63	907.20	400.74	527.49	428.28	943.04
Min-Max (OS)	149.69	149.69	149.69	149.69	149.69	149.69	149.69
Min-Max (OS)	149.94	149.94	151.36	149.94	149.94	149.94	149.94
Min-Max (OS)	149.94	149.94	151.36	149.94	149.94	149.94	149.94
GA (Binary)	2.83	307.47	2.97	1.85	164.10	0.33	0.25
GA (Binary)	999.99	999.99	999.99	999.99	999.99	0.40	408.46
GA (Binary)	999.99	999.99	999.99	999.99	999.99	983.40	990.65
GA (FP)	0.25	163.06	25.26	55.34	40.91	0.25	0.25
GA (FP)	999.68	995.99	999.99	969.99	999.99	334.54	11.04
GA (FP)	999.78	999.98	999.99	999.97	999.90	999.90	999.79
Literature	0.40	61.50	93.00	38.90	47.80	0.41	0.42
Literature	0.40	61.50	93.00	38.90	47.80	0.41	0.42
Literature	0.40	61.50	93.00	38.90	47.80	0.41	0.42

Table 6.9: (Part II) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.10 and 6.11)

Method	$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$	$x_{15}$	$x_{16}$	$x_{17}$
Monte Carlo 1	336.86	247.97	277.32	165.75	474.50	80.25	154.58
Monte Carlo 1	50.69	610.30	148.90	758.23	335.02	880.26	549.24
Monte Carlo 1	50.69	610.30	148.90	758.23	335.02	880.26	549.24
Min-Max (OS)	149.69	149.69	149.69	149.69	149.69	149.69	149.69
Min-Max (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
Min-Max (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
GA (Binary)	10.57	0.27	0.87	41.47	170.62	6.78	163.87
GA (Binary)	999.99	685.44	999.99	999.99	999.99	999.99	999.99
GA (Binary)	999.99	974.16	999.99	999.99	999.99	999.99	999.99
GA (FP)	1.44	1.07	74.05	0.25	40.98	75.85	200.27
GA (FP)	0.55	1.21	999.99	999.99	999.98	999.99	999.99
GA (FP)	999.74	999.97	999.99	999.99	999.96	999.99	999.95
Literature	7.07	0.40	27.10	24.20	43.20	73.40	50.10
Literature	7.07	0.40	27.10	24.20	43.20	73.40	50.10
Literature	7.07	0.40	27.10	24.20	43.20	73.40	50.10

Table 6.10: (Part III) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.11)

Method	$x_{18}$	$x_{19}$	$x_{20}$
Monte Carlo 1	149.35	202.75	120.33
Monte Carlo 1	919.97	817.87	337.84
Monte Carlo 1	919.97	817.87	337.84
Min-Max (OS)	149.69	149.69	149.69
Min-Max (OS)	149.94	149.94	149.94
Min-Max (OS)	149.94	149.94	149.94
GA (Binary)	223.78	4.60	0.49
GA (Binary)	999.99	999.99	999.99
GA (Binary)	999.99	999.99	999.99
GA (FP)	200.32	0.25	0.25
GA (FP)	986.32	999.99	999.94
GA (FP)	504.66	999.99	999.84
Literature	70.10	1.00	2.60
Literature	70.10	1.00	2.60
Literature	70.10	1.00	2.60

Table 6.11: (Part IV) Comparison of results computing the ideal vector of example 4 from Chapter 5 (design of a 10-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System.

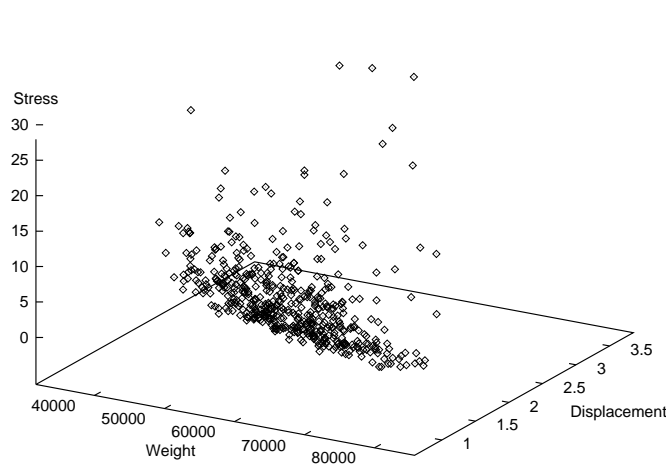


Figure 6.84: Initial feasible region for Monte Carlo method solving the fourth example.

considering increments of 0.01 we would have to perform  $100000^{20}$  iterations to generate the feasible zone (about  $1 \times 10^{100}$  iterations). Even if we consider only 100 values from each range, we have to generate  $1 \times 10^{40}$  points. However, I will display the 3-dimensional representation of the objective functions when each method is applied to this example, and this will give us a better idea of how the feasible zone looks like, although it may be harder to identify the Pareto front from such graphs. To solve this problem, it was necessary to add a module to each program in order to analyze the plane truss generated by each algorithm. This module uses the matrix factorization method included in Gere and Weaver [202] together with the stiffness method [202] [203] to analyze the structure.

The ideal vector of this problem was computed using Monte Carlo Methods 1 and 2 (generating 500 points) presented in Chapter 4, and a GA (with a population of 500 chromosomes running during 100 generations) using binary and floating point representation, with the procedure described in Chapter 4 to adjust its parameters. The corresponding results are shown in Tables 6.8, 6.9, 6.10 and



6.11 including the best results reported in the literature [193]. This time, there is a great difference between the results found by the GA and the mathematical programming techniques, confirming two hypothesis about them: a) Monte Carlo methods are not very reliable when the search space is too large, as in this case, and b) Mathematical Programming techniques need a very good guess point to start the search, and if this value is not close to the optimum region (as in this case) the results tend to be completely undesirable. Notice that, as in previous cases, the GA with floating point representation produced the best results, except for the last objective, for which the binary representation found a slightly better solution. However, again, we can say that, in general, the GA with floating point representation produced the best results (i.e., the ideal vector). Notice that the set of results reported by Belegundu [193] was produced optimizing only the first objective (i.e., the total weight of the truss). We can see how the GA with floating point representation found a better result than that reported by Belegundu, showing, once more, that this technique can overcome mathematical programming techniques when used properly (i.e., if we can find the right set of parameters, which is what the procedure described in Chapter 4 does). The results for Monte Carlo Method 2 are the same than for Method 1.

Since the Monte Carlo Methods previously mentioned and Osyczka's multiobjective optimization system do not generate the Pareto front, I will compare them with the GA-based approaches only in terms of the best overall result found. Besides those results, it is interesting to observe the initial distribution of points randomly generated by the method (see Figure 6.84). As can be seen from the tables previously shown (see Tables 6.8, 6.9, 6.10 and 6.11), the objectives are highly conflicting, and therefore, it is very hard to come up with a good compromise. From the graph depicted in Figure 6.84 we can see that the optimum region

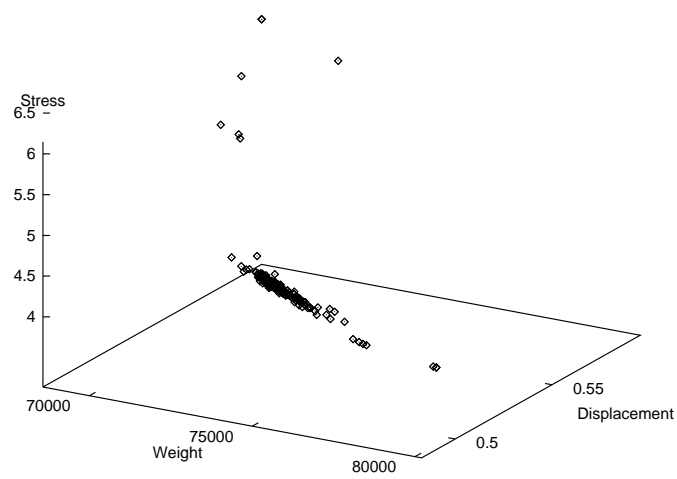


Figure 6.85: Example 4: The GA using a linear combination of the objectives with scaling, after 100 generations using binary representation.

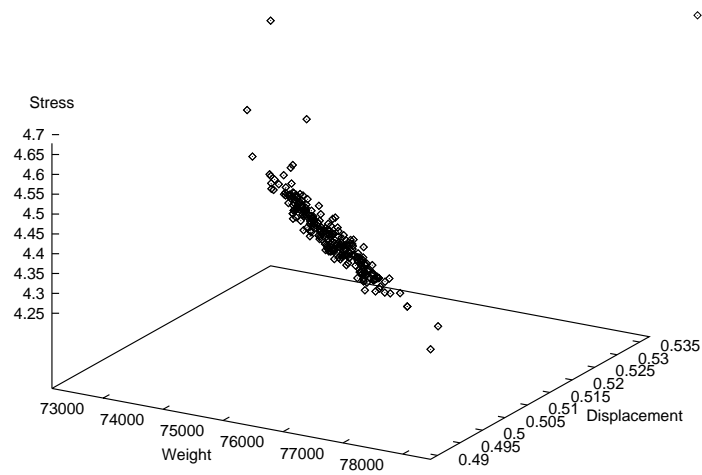


Figure 6.86: Example 4: The GA using a linear combination of the objectives with scaling, after 100 generations using floating point representation.

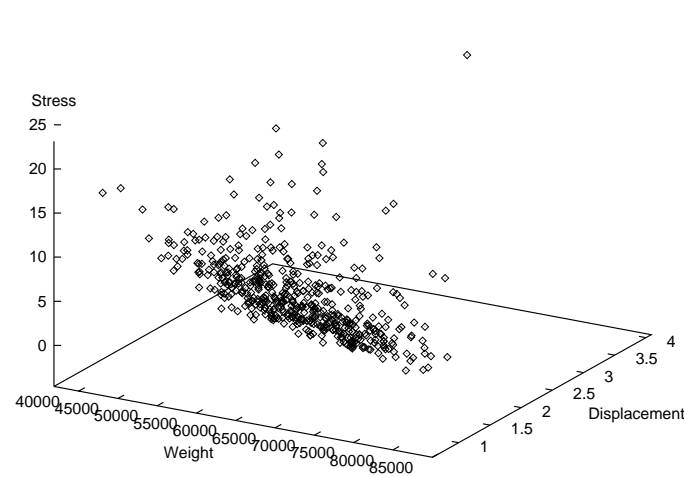


Figure 6.87: Example 4: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed.

would be located as close as possible to the ideal vector for each one of the axis. However, because of the great variation in the results, we will see that the common compromise will be to have higher values of weight to allow smaller stresses and displacements. This is true in the physical world, since larger cross-sectional areas allow less stress and displacement in the structure. It is also important observe through all the graphs corresponding to the example the ranges of the axis, because even when the clusters of points could all look alike, there will be a great variation in terms of the ranges. For example, the initial region for Monte Carlo Methods presents a trend towards relatively low weights and stresses and higher displacements.

We will start by analyzing the behavior of the **Lexicographic Method** at different stages of the search process. Figure 6.87 shows the distribution of points produced at generation zero using this method. We can see that this distribution has many similarities with the initial distribution of Monte Carlo methods, previously shown, except that in this case the ranges are smaller and there seems

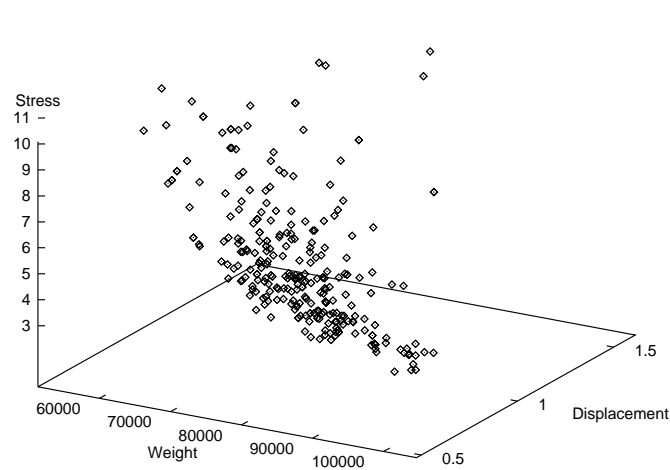


Figure 6.88: Example 4: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed.

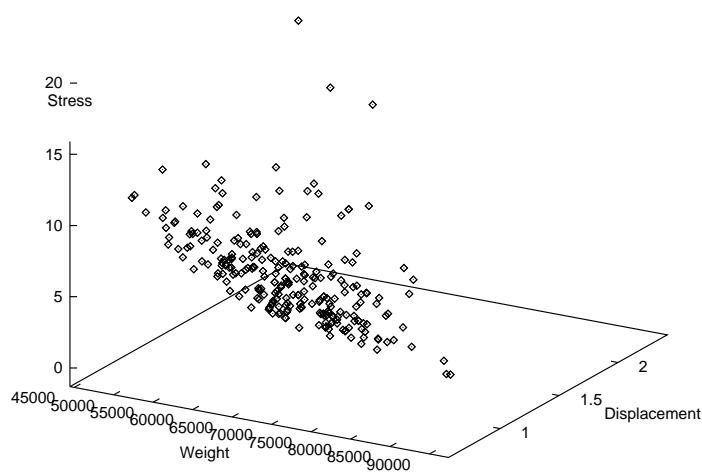


Figure 6.89: Example 4: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed.

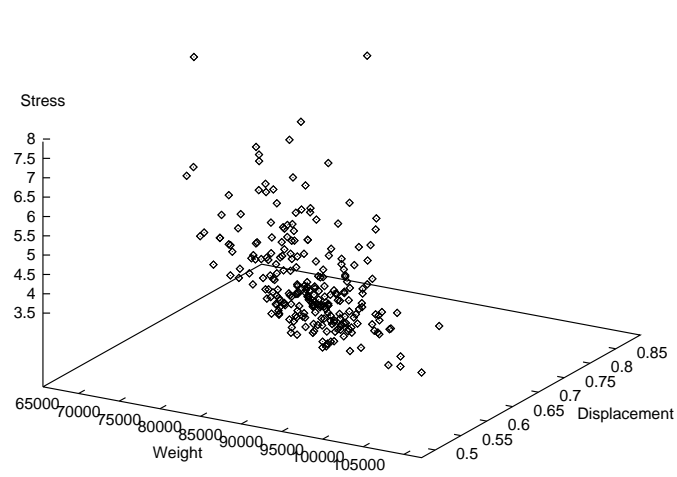


Figure 6.90: Example 4: Distribution of points using the Lexicographic Method with a binary representation at generation 100. Only points within the feasible region are displayed.

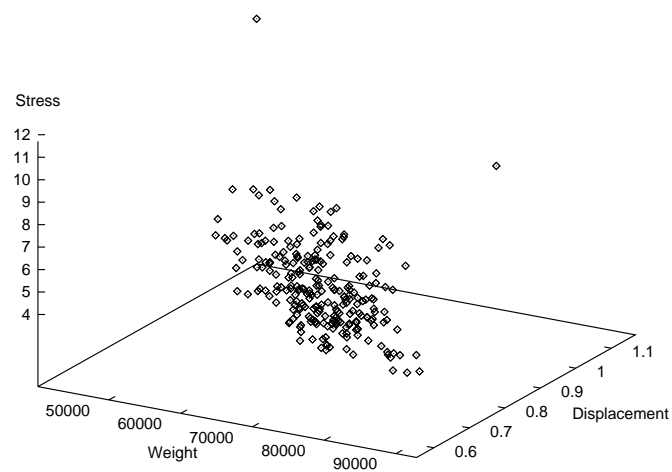


Figure 6.91: Example 4: Distribution of points using the Lexicographic Method with a floating point representation at generation 100. Only points within the feasible region are displayed.

to be a slightly better distribution of points. Figure 6.88 shows the distribution of points at generation 20 using binary representation, and Figure 6.89 shows the corresponding distribution using floating point representation. As in previous cases, we can see that using a floating point representation up to this stage of the search process, we can get a more uniform distribution of points, with fewer dominated points than when we use binary representation, and with compromises that favor weight over stress and displacement. After 100 generations, floating point representation presents points more uniformly distributed, almost shaping a plane that seems to be the Pareto front (see Figure 6.91). As mentioned before, the compromise taken was to reduce weight, sacrificing displacement and stress, whereas in the solution found using binary representation (see Figure 6.90) there is more dispersion of the points, and the compromise seems to be the reduction of stress and displacement, sacrificing weight. This is a good example to illustrate the importance of the scheme representation chosen to apply a genetic algorithm. In this case, the use of floating point representation provided a more compact distribution of points and a better overall solution (see Tables 6.12, 6.13, 6.14) and 6.15), but it also could seem more practical to sacrifice one objective (weight, in this example) to improve the other two (displacement and stress) rather than what this technique did (exactly the opposite). I also compared the effect of a simple linear combination of objectives (addition or multiplication) using scaling, and the results after 100 generations are shown in Figure 6.85. Observe how in this case, even a good scaling of the objectives is not able to find a good trade-off, and, as expected, is also unable to delineate the Pareto front. The population starts converging to a small set of points in which weight and displacement are favored, and stress is sacrificed. The use of floating point representation gives a very similar distribution, and can not avoid global convergence even when this is

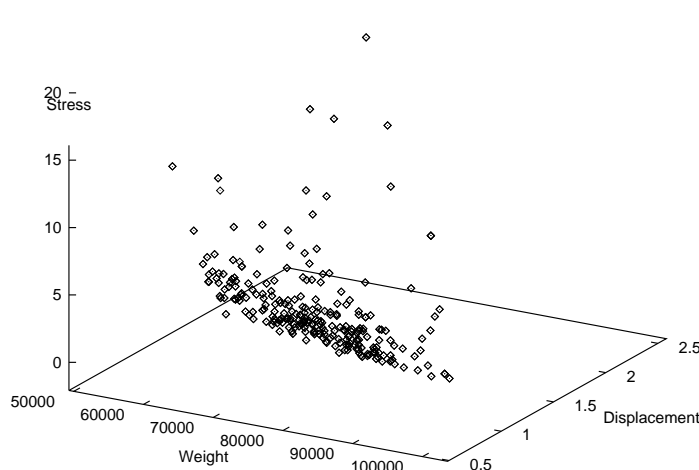


Figure 6.92: Example 4: Distribution of points using VEGA with a binary representation at generation twenty.

not totally evident in the graph until we see the ranges in the axis (see Figure 6.86). In this case, however, stress and displacement are given priority over weight. The best overall result in both cases is very poor, with the compromise established by floating point representation being worse in terms of the ideal vector (although the decision maker could still want a compromise of that sort).

Schaffer's VEGA produces a reasonably good distribution of points after 20 generations using binary representation (see Figure 6.92), but it still has several dominated points within the population. Up to this point, weight has priority over stress and displacement. After 50 generations, however, things seem to change, and now the compromise is to favor displacement and stress, sacrificing weight (see Figure 6.93). At this stage, there are fewer dominated points in the graph, and we can start visualizing a trade-off contour. After 100 generations, we have partial convergence of the population towards a region that favors displacement and stress over weight, with only a few points escaping from this region (see Figure 6.96). The best overall solution produced using this method with binary

Method	$x_1$	$x_2$	$f_1$	$f_2$	$f_3$	$L_p(f)$
Ideal Vector			4638.27	0.441966	3.333183	0.000000
Monte Carlo 1	568.79	548.29	53192.31	0.965460	7.648280	12.94718
Monte Carlo 2	739.06	862.57	55219.31	0.855851	7.103715	12.97281
Min-max (OS)	149.94	149.94	18868.75	2.948609	22.237597	14.41121
GCM (OS)	149.94	149.94	18878.93	2.942967	22.206487	14.39130
WMM (OS)	149.94	149.94	18868.75	2.948609	22.237597	14.41121
PMM (OS)	149.69	149.69	18836.86	2.953601	22.275247	14.42692
NMM (OS)	149.94	149.94	18878.93	2.942967	22.206487	14.39130
GALC (B)	999.99	999.99	69569.32	0.557806	5.429589	14.89001
GALC (FP)	986.23	996.95	72453.42	0.527797	4.521338	15.17143
Lexicographic (B)	211.19	815.41	60622.20	0.766283	7.931645	14.18339
Lexicographic (FP)	445.21	922.61	40120.69	1.167479	11.700044	11.80164
VEGA (B)	40.42	273.03	57619.82	1.379811	9.670593	15.44597
VEGA (FP)	988.71	820.67	67208.04	0.676467	6.405205	14.94211
NSGA (B)	677.02	999.99	88579.33	0.532519	4.648191	18.69688
NSGA (FP)	871.94	877.51	83412.55	0.567590	4.614603	17.65221
MOGA (B)	999.99	999.99	48199.54	0.964701	7.932508	11.95430
MOGA (FP)	972.31	762.56	43846.84	1.007271	8.380988	11.24674
NPGA (B)	999.99	275.11	62843.73	0.874356	8.118541	14.96295
NPGA (FP)	14.40	57.03	50687.80	0.840845	12.244041	13.50404
Hajela (B)	812.58	81.82	51259.10	1.44721	9.118564	13.37708
Hajela (FP)	262.96	433.37	46976.37	1.319407	9.631833	13.00298
GAminmax1 (B)	999.99	349.50	29984.80	1.183646	10.203125	9.20386
GAminmax1 (FP)	597.63	484.86	28053.57	1.290751	10.923750	9.24603
GAminmax2 (B)	999.99	999.99	112587.49	0.441966	3.698968	23.38331
GAminmax2 (FP)	995.78	999.25	108346.09	0.448540	3.716841	22.48911

Table 6.12: (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.13, 6.14 and 6.15)



Method	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$
Monte Carlo 1	82.46	713.22	273.77	686.78	478.02	389.66	711.35
Monte Carlo 2	27.50	419.06	314.71	560.84	875.14	242.53	422.33
Min-max (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
GCM (OS)	149.94	149.94	149.94	151.36	149.94	149.94	149.94
WMM (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
PMM (OS)	149.69	149.69	149.69	149.69	149.69	149.69	149.69
NMM (OS)	149.94	149.94	149.94	151.36	149.94	149.94	149.94
GALC (B)	453.70	92.95	999.99	999.99	999.99	687.66	2.49
GALC (FP)	443.74	105.54	973.44	990.69	698.35	853.29	7.56
Lexicographic (B)	103.79	680.00	999.99	625.02	413.83	943.66	96.33
Lexicographic (FP)	119.93	705.79	982.58	531.84	63.94	182.92	212.88
VEGA (B)	747.28	227.40	999.99	43.95	754.28	999.99	904.53
VEGA (FP)	330.54	642.00	559.35	934.17	871.85	772.50	276.30
NSGA (B)	999.99	932.40	999.99	999.99	539.62	866.29	999.99
NSGA (FP)	571.91	586.58	817.15	696.57	275.67	770.45	832.84
MOGA (B)	595.63	999.99	872.65	778.18	976.24	999.99	993.94
MOGA (FP)	420.26	216.58	630.09	937.91	596.98	329.61	135.19
NPGA (B)	144.28	761.98	362.95	730.09	513.68	946.59	3.00
NPGA (FP)	913.50	293.54	942.89	967.59	897.88	663.62	746.60
Hajela (B)	677.37	119.94	999.99	306.19	999.99	994.26	485.06
Hajela (FP)	915.82	234.17	563.21	192.44	712.29	428.97	163.35
GAminmax1 (B)	41.69	36.41	739.40	300.38	775.73	81.50	19.83
GAminmax1 (FP)	139.44	22.04	353.38	418.54	327.86	316.09	56.63
GAminmax2 (B)	999.99	999.99	999.99	999.99	999.99	999.99	182.66
GAminmax2 (FP)	994.69	935.50	995.36	996.86	994.25	994.82	10.11

Table 6.13: (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.14 and 6.15)

Method	x <sub>10</sub>	x <sub>11</sub>	x <sub>12</sub>	x <sub>13</sub>	x <sub>14</sub>	x <sub>15</sub>	x <sub>16</sub>
Monte Carlo 1	420.06	179.62	548.32	49.21	454.42	100.22	572.62
Monte Carlo 2	262.27	403.24	519.37	969.49	317.29	763.80	243.18
Min-max (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
GCM (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
WMM (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
PMM (OS)	149.69	149.69	149.69	149.69	149.69	149.69	149.69
NMM (OS)	149.94	149.94	149.94	149.94	149.94	149.94	149.94
GALC (B)	1.65	244.40	44.11	999.99	999.99	29.56	999.99
GALC (FP)	10.86	262.61	7.76	975.55	950.37	881.98	698.79
Lexicographic (B)	999.99	356.46	999.99	586.39	998.94	321.76	593.05
Lexicographic (FP)	6.81	109.10	586.41	43.80	956.88	712.47	430.76
VEGA (B)	734.12	101.46	138.12	909.71	359.34	495.71	999.99
VEGA (FP)	584.72	222.63	436.98	570.08	791.96	871.64	946.63
NSGA (B)	878.05	735.83	568.36	999.99	788.23	749.83	999.99
NSGA (FP)	521.01	946.13	849.67	981.95	965.13	488.56	902.68
MOGA (B)	999.99	407.08	94.87	170.13	999.99	846.14	414.63
MOGA (FP)	704.39	811.73	888.41	143.05	591.59	981.12	571.07
NPGA (B)	538.34	999.99	11.88	218.09	463.96	950.37	999.99
NPGA (FP)	621.74	767.30	689.26	445.81	719.91	670.87	865.13
Hajela (B)	37.71	999.99	999.99	999.99	363.73	375.61	708.13
Hajela (FP)	873.88	577.44	280.74	430.82	876.98	183.82	391.43
GAminmax1 (B)	4.82	40.87	7.89	778.72	102.08	856.51	160.79
GAminmax1 (FP)	9.78	79.04	11.61	209.11	336.30	222.86	266.77
GAminmax2 (B)	478.74	181.67	999.99	999.99	999.99	999.99	999.99
GAminmax2 (FP)	127.64	267.33	432.67	966.51	958.36	996.60	998.63

Table 6.14: (Part III) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Table 6.15)

Method	$x_{17}$	$x_{18}$	$x_{19}$	$x_{20}$
Monte Carlo 1	412.64	228.87	484.10	41.59
Monte Carlo 2	822.99	290.74	124.14	388.45
Min-max (OS)	149.94	151.36	149.94	149.94
GCM (OS)	149.94	149.94	149.94	149.94
WMM (OS)	149.94	151.36	149.94	149.94
PMM (OS)	149.69	149.69	149.49	149.69
NMM (OS)	149.94	149.94	149.94	149.94
GALC (B)	999.99	939.14	612.52	3.38
GALC (FP)	991.55	353.91	715.32	32.00
Lexicographic (B)	836.76	999.99	944.36	373.50
Lexicographic (FP)	659.13	299.28	908.48	568.38
VEGA (B)	999.99	161.84	0.61	133.48
VEGA (FP)	909.70	130.85	422.78	718.08
NSGA (B)	453.50	679.71	999.99	713.62
NSGA (FP)	982.48	895.44	946.93	584.22
MOGA (B)	999.99	578.93	297.43	999.99
MOGA (FP)	389.85	708.47	464.30	620.24
NPGA (B)	54.88	309.74	152.27	491.54
NPGA (FP)	961.66	385.32	740.64	516.32
Hajela (B)	999.99	647.67	705.56	163.89
Hajela (FP)	993.84	155.72	631.91	53.29
GAminmax1 (B)	999.99	798.45	6.99	3.94
GAminmax1 (FP)	844.37	376.55	49.98	9.64
GAminmax2 (B)	999.99	902.14	999.99	999.99
GAminmax2 (FP)	933.07	856.75	995.59	981.88

Table 6.15: (Part IV) Comparison of the best overall solution found by each one of the methods included in MOSES for the fourth example (design of a 10-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective).

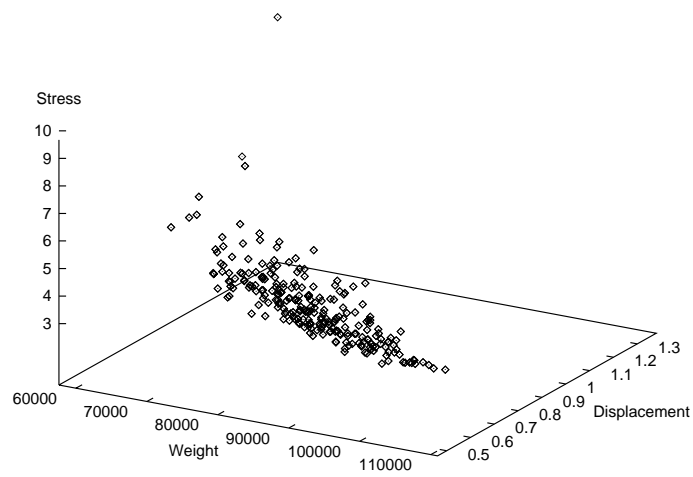


Figure 6.93: Example 4: Distribution of points using VEGA with a binary representation at generation fifty.

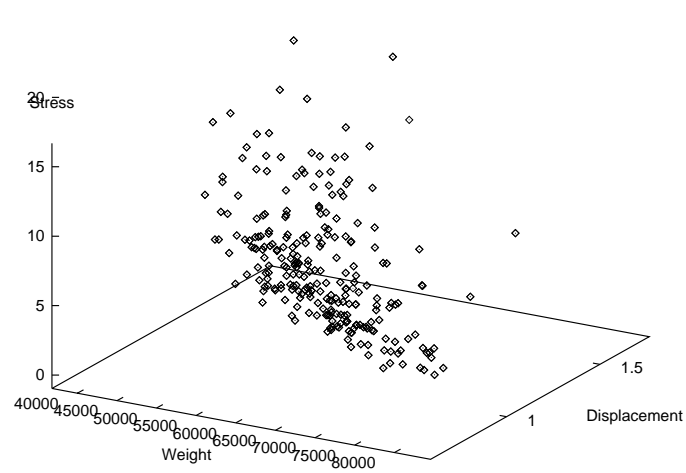


Figure 6.94: Example 4: Distribution of points using VEGA with floating point representation at generation twenty.

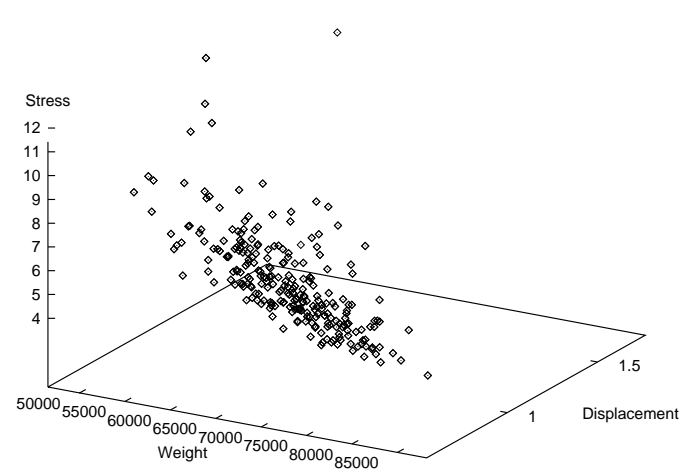


Figure 6.95: Example 4: Distribution of points using VEGA with floating point representation at generation fifty.

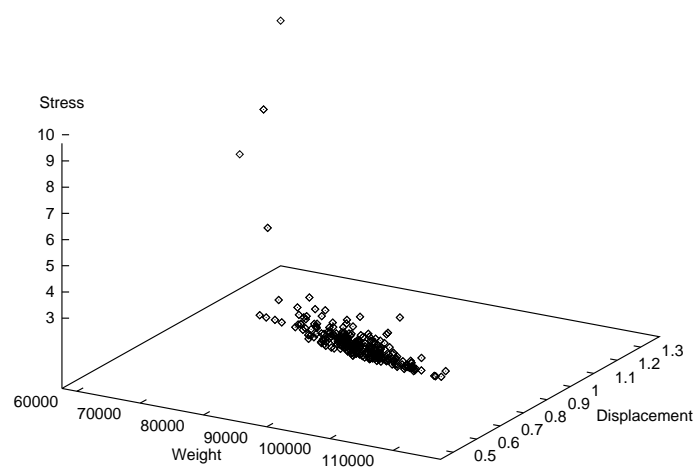


Figure 6.96: Example 4: Distribution of points using VEGA with binary representation at generation 100.

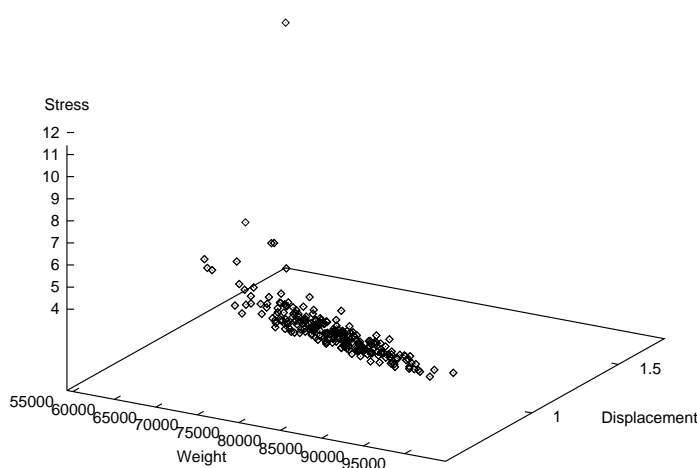


Figure 6.97: Example 4: Distribution of points using VEGA with floating point representation at generation 100.

representation is not very good, because of its trend to favor individuals who excel in only one dimension, rather than to keep the best trade-offs. After only 20 generations, floating point representation seems to be doing better in terms of obtaining compromises, but has more problems to keep the distribution of points uniform (see Figure 6.94). After 50 generations, stress gains some importance over weight, and the final contour starts to appear (see Figure 6.95). Finally, after 100 generations, the population seems to be converging towards the best compromise found by the algorithm, favoring stress and displacement over weight. Notice that the weight has increased somewhat more than before, but the stress has decreased (see Figure 6.97). Floating point representation was able to find a better overall solution than binary representation in this example, by allowing a slightly higher displacement and stress in order to reduce the total weight of the structure. As in previous examples, global convergence to a single point seems to be just a matter of time when using this technique.

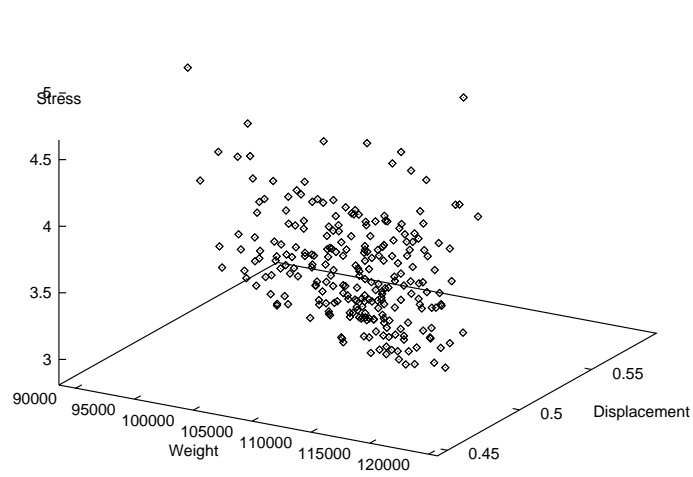


Figure 6.98: Example 4: Distribution of points using NSGA with binary representation at generation 100.

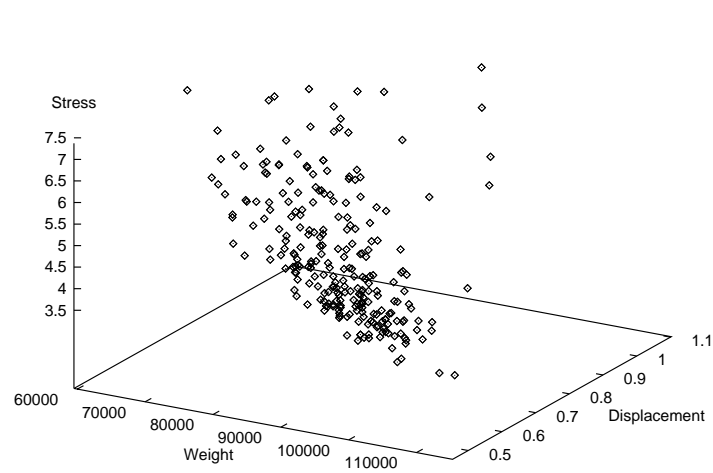


Figure 6.99: Example 4: Distribution of points using NSGA with binary representation at generation twenty.

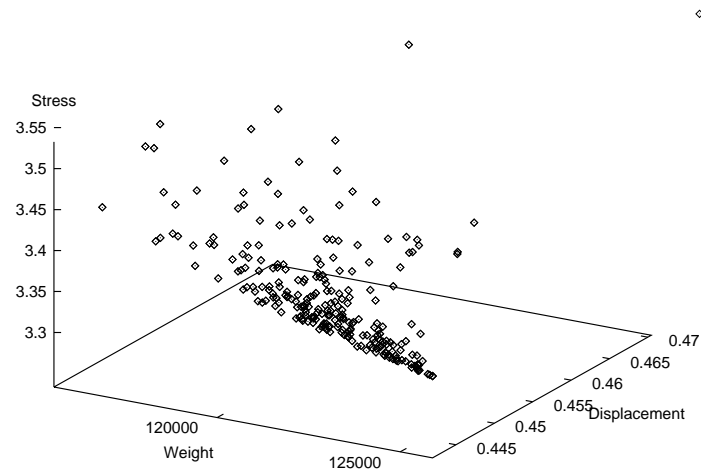


Figure 6.100: Example 4: Distribution of points using NSGA with binary representation at generation 500.

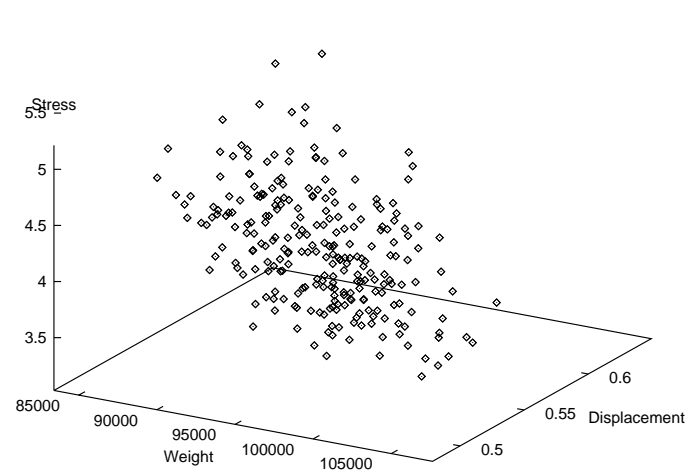


Figure 6.101: Example 4: Distribution of points using NSGA with floating point representation at generation 100.



After only 20 generations, my version of Srinivas' NSGA that uses binary representation presents a slightly compact cluster of points that represent compromises in which weight has priority over displacement and stress (see Figure 6.99). However, after 100 generations, we can see that the compromise is now to prioritize stress and displacement over weight, and the points are distributed in a lesser compact way than before (see Figure 6.98). After 500 generations, the trend is to make the population converge towards a solution in which stress and displacement are extremely close to the ideal vector, but the weight is excessive (see Figure 6.100). The selection mechanism of this technique seems to favor solutions that highly dominate with respect to those two objectives, sacrificing compromises that would balance better with respect to weight. Again, to converge to a global trade-off with two almost ideal objectives seems to be only a matter of time. If we use floating point representation, we can anticipate the previous results, since after 100 generations there seems to be a strong trend towards solutions in which stress and displacement take precedence over weight (see Figure 6.101), however, since the points are distributed over a slightly larger range, this representation produces a better overall solution up to this point. Once more, there was not significant change in the results when the value of  $\sigma_{share}$  was modified.

Fonseca's MOGA gives a good region of compromise solutions after 20 generations using binary representation, favoring weight over displacement and stress (see Figure 6.102). After 100 generations, the behavior of the technique is very consistent, since now we have less dominated points, and the compromise solutions have a slightly lower stress than before, but we still have a relatively low weight (see Figure 6.103). Due to the good compromise solutions present at this stage of the search, the best overall result reported at this point is remarkably good. After 500 generations, the method shows a set of points less compact than

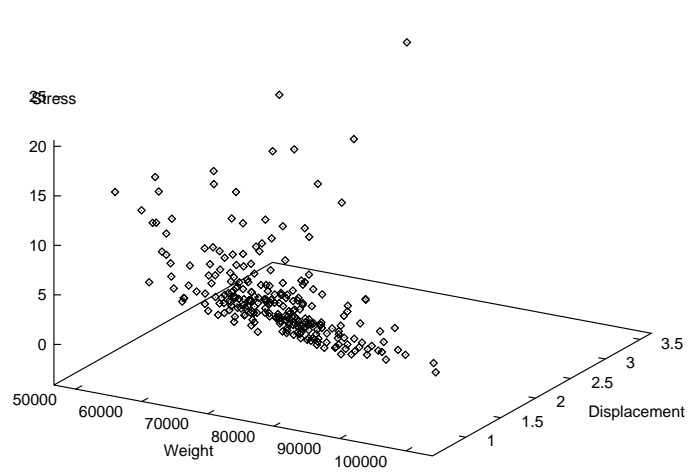


Figure 6.102: Example 4: Distribution of points using MOGA with binary representation at generation 20.

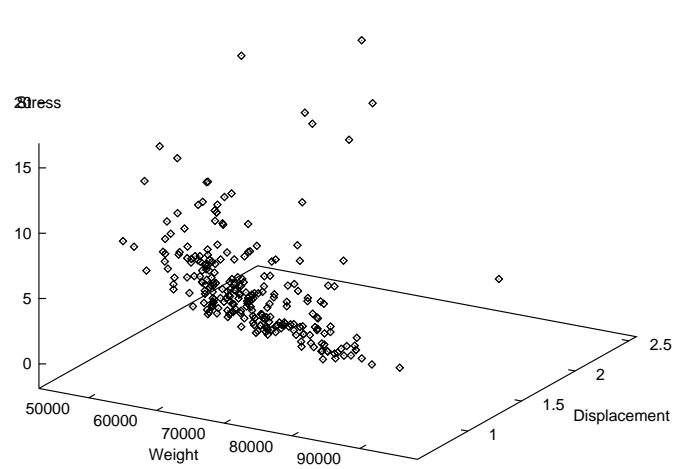


Figure 6.103: Example 4: Distribution of points using MOGA with binary representation at generation 100.

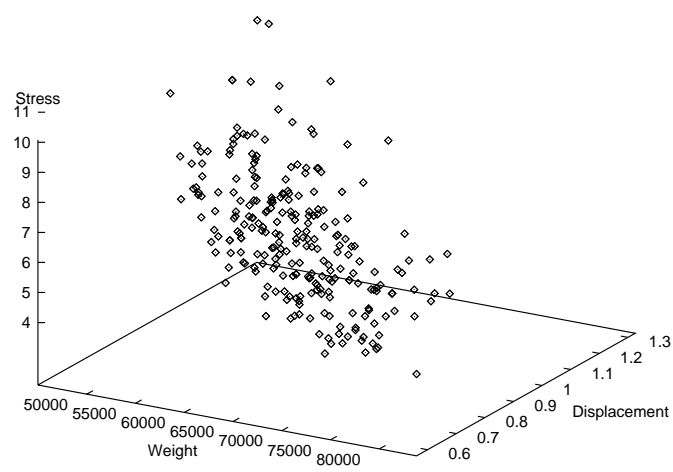


Figure 6.104: Example 4: Distribution of points using MOGA with binary representation at generation 500.

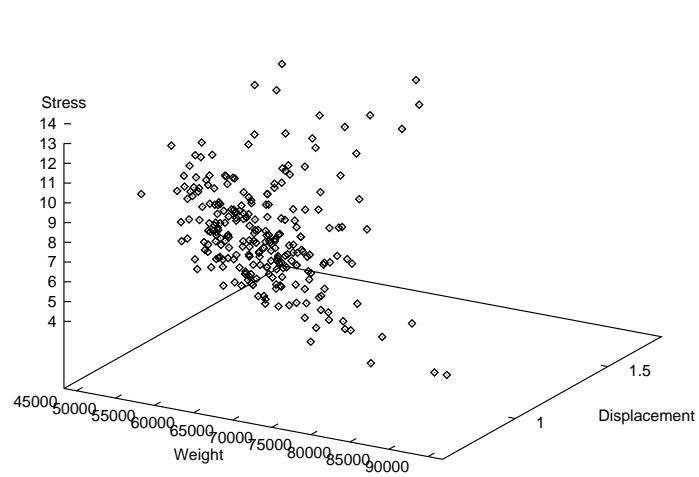


Figure 6.105: Example 4: Distribution of points using MOGA with floating point representation at generation 100.

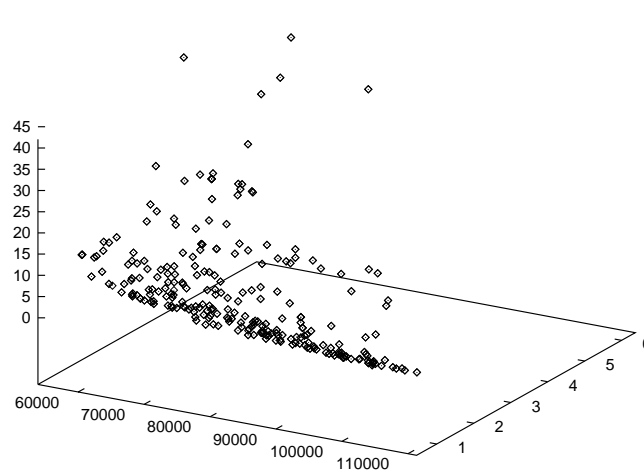


Figure 6.106: Example 4: Distribution of points using NPGA with binary representation at generation 20.

at earlier stages in the search, with better compromise solutions (i.e., solutions that favor the three objectives), but without clear indication of where the Pareto front is (see Figure 6.104). The best overall solution found by this technique is excellent using both representations, and the one produced using floating point representation was the best found. The graph after 100 generations using floating point representation shows for this problem more sparsely distributed points than when using binary representation, but the compromises are better since the ranges of the objectives are smaller (see Figure 6.105). I also used a value of  $\sigma_{share} = 0.1$  as in NSGA, but changing this value did not affect the results produced by the algorithm.

After 20 generations, NPGA produces a quite compact cluster of points, but there is a large variation in the ranges of them (see Figure 6.106), and we can clearly see a lot of dominated points. After 50 generations, we still have a large variation in the ranges of the objectives (see Figure 6.107). At generation 100, the trend seems clear: weight is being favored over displacement and stress

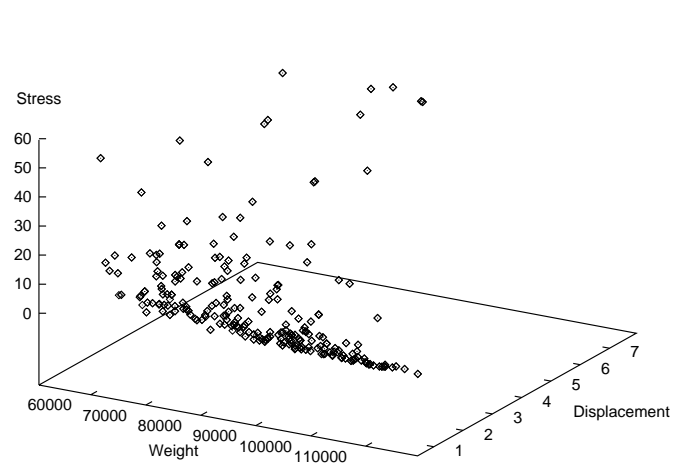


Figure 6.107: Example 4: Distribution of points using NPGA with binary representation at generation 50.

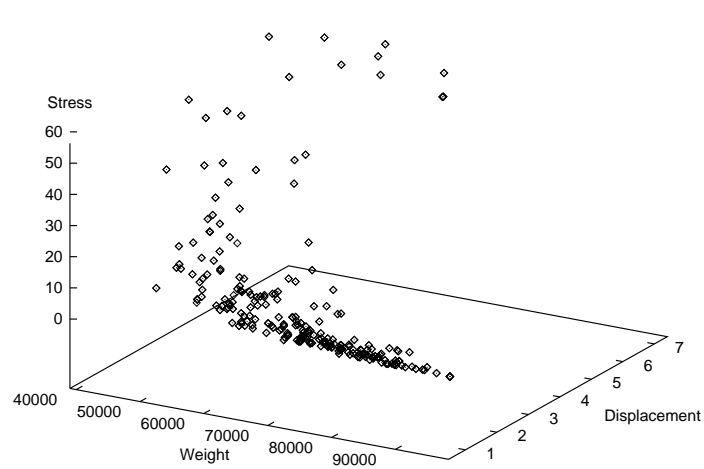


Figure 6.108: Example 4: Distribution of points using NPGA with binary representation at generation 100.

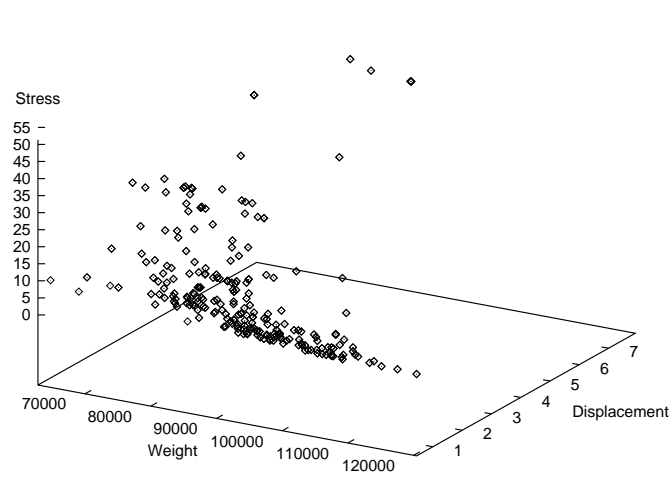


Figure 6.109: Example 4: Distribution of points using NPGA with binary representation at generation 500.

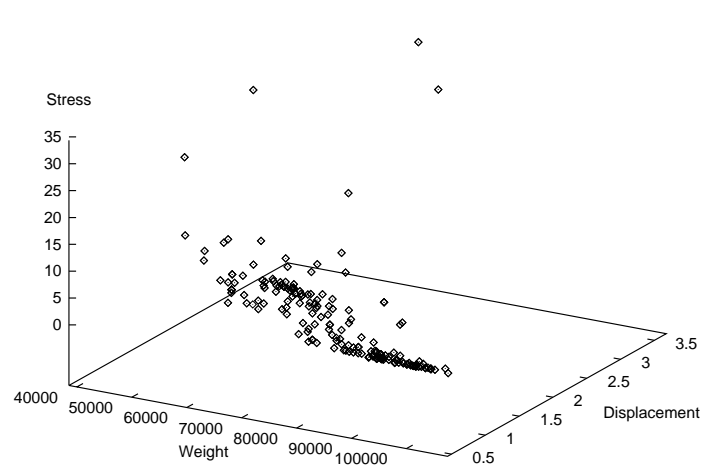


Figure 6.110: Example 4: Distribution of points using NPGA with binary representation at generation 100 with  $\sigma_{share} = 1.0$ .

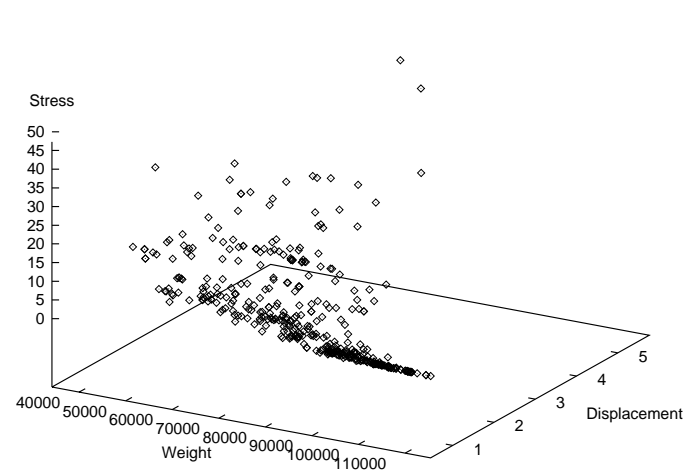


Figure 6.111: Example 4: Distribution of points using NPGA with floating point representation at generation 100.

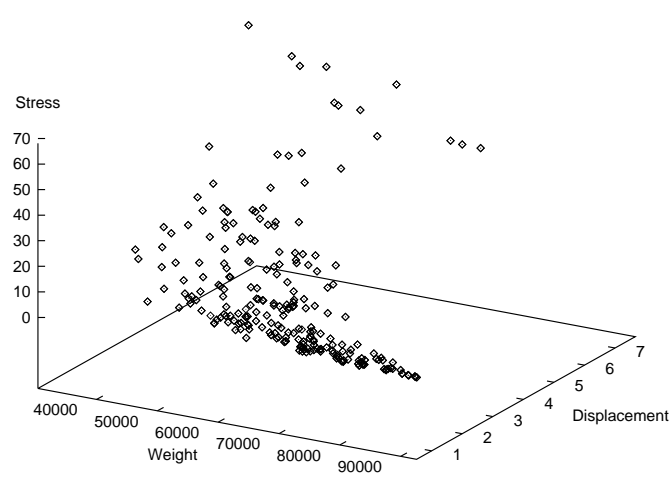


Figure 6.112: Example 4: Distribution of points using NPGA with floating point representation at generation 20.

(see Figure 6.108). As in previous cases, we can see that this method does not converge to a single solution and even after 500 generations, the population looks very well distributed, but the stress has taken some priority over weight, producing slightly better trade-offs than before (see Figure 6.109). As in previous examples, this method was able to keep more points through a lot of generations than any other Pareto-based technique. The best overall solution produced when binary representation was used scores average compared to the other techniques, and improves a little if we use floating point representation. As in previous examples, the value of  $\sigma_{share}$  plays a critical role in the performance of the algorithm, because if we increase it from 0.1 (same value used with the previous techniques that use niches) to 1.0, the distribution of points is less compact than before, slightly favoring displacement over weight (see Figure 6.110). This trade-off allows the technique to find a slightly better overall solution, but more infeasible points are generated in the graph. The value of  $t_{dom}$  significantly affects the performance of the algorithm both in terms of efficiency (it gets much slower) and quality of results (it could completely fail in finding the Pareto front). If we use floating point representation in this example, we will be able to see, even in early stages of the search, that weight is being given priority over displacement and stress (see Figure 6.112). After 100 generations, the technique keeps consistency, and there is a quite compact cluster of points of compromise solutions in which stress was slightly favored over weight, and from which we may extract an overall solution which is better than that produced with binary representation (see Figure 6.111).

When we use Hajela's method, the results seem to be more consistent than before, probably because I chose a good set of weights. After 20 generations, there is a more or less uniform distribution of points, encompassing compromise solutions in which stress and displacement have priority over weight (see Figure 6.113).



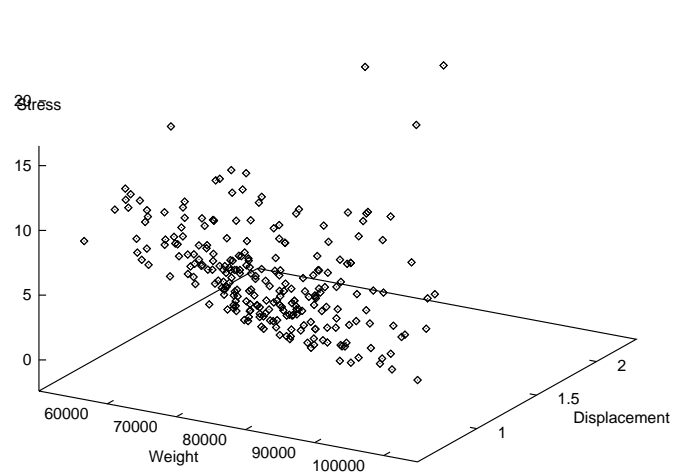


Figure 6.113: Example 4: Distribution of points using Hajela's method with binary representation at generation 20.

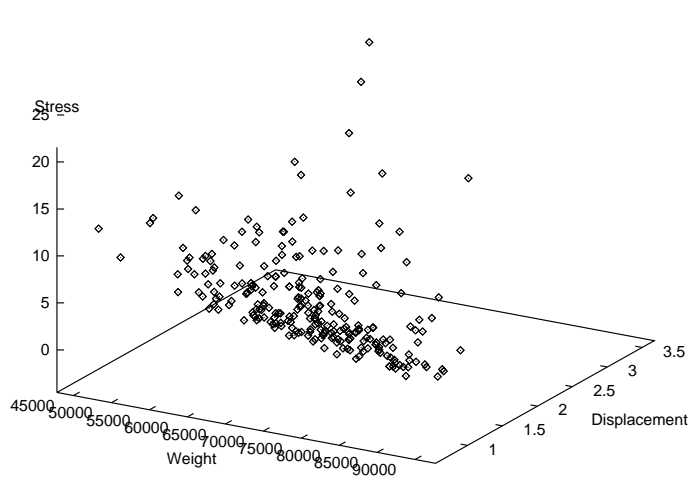


Figure 6.114: Example 4: Distribution of points using Hajela's method with binary representation at generation 50.

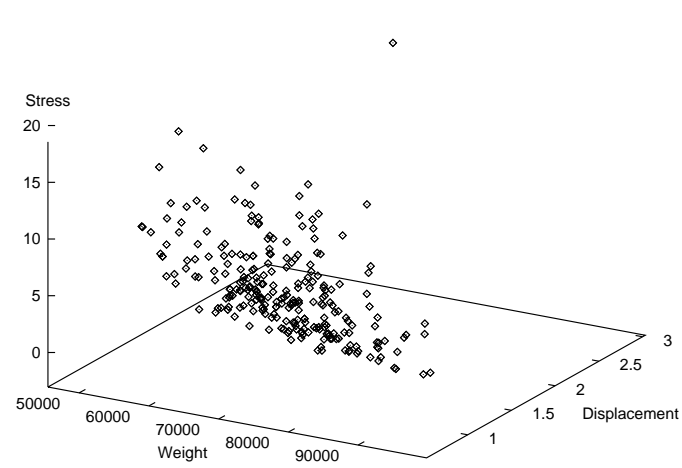


Figure 6.115: Example 4: Distribution of points using Hajela's method with binary representation at generation 100.

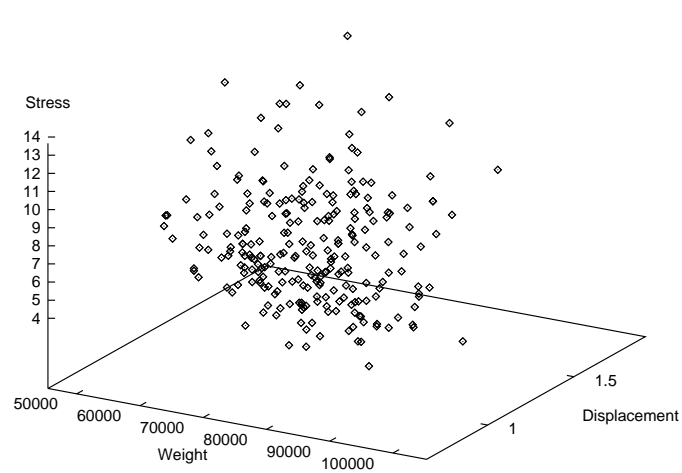


Figure 6.116: Example 4: Distribution of points using Hajela's method with binary representation at generation 500

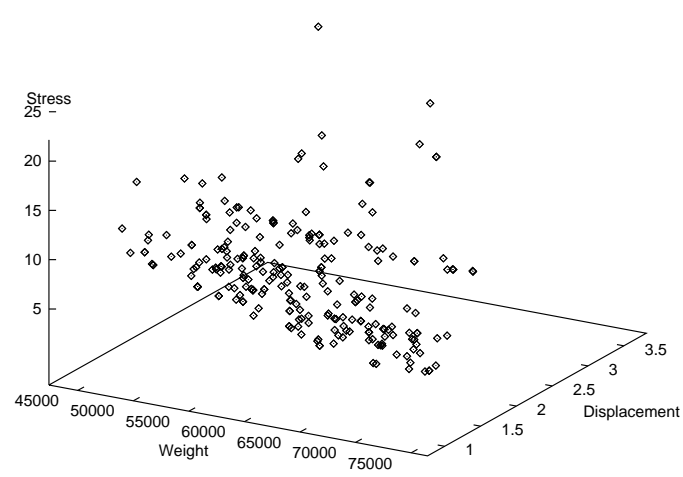


Figure 6.117: Example 4: Distribution of points using Hajela's method with floating point representation at generation 20.

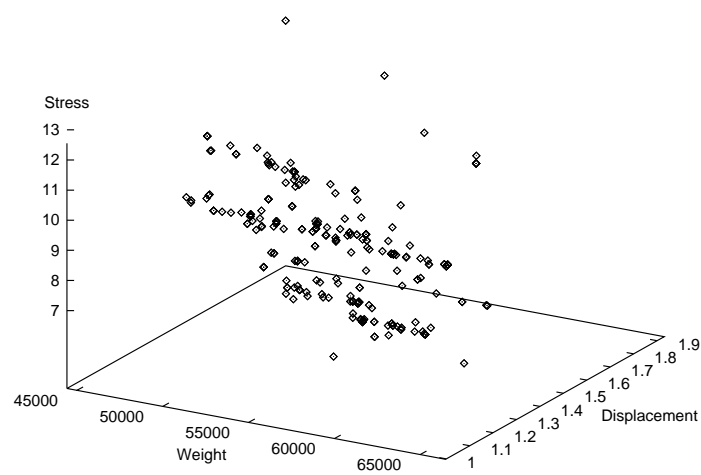


Figure 6.118: Example 4: Distribution of points using Hajela's method with floating point representation at generation 100.

After 50 generations weight is given a higher priority, and displacement and stress values get increased (see Figure 6.114). At generation 100, the technique has found very good compromises, as the corresponding graph shows (see Figure 6.115). The best overall results found by this technique using both representations are above average, mainly because of the fairly large range of the objective values. To see if the technique could keep the diversity of the population, I ran the algorithm for 500 generations, and although the technique loses the Pareto front, the diversity of the population is maintained (see Figure 6.116). The trade-offs after that many generations favor displacement and stress rather than weight, but no main convergence trend can be observed in the population. The use of floating point representation provides a more solid Pareto front after 20 generations, with a trade-off that favors stress and displacement over weight (see Figure 6.117). However, after 100 generations the front has been lost, and the population looks quite sparse, with a compromise that favors stress and weight over displacement (see Figure 6.118). The diversity of such population explains that we are able to find a better overall solution when floating point representation is used.

Before showing the results produced with my method based on the min-max approach, I want to show the distribution of points at generation zero, since in this case the algorithm ensures that only feasible solutions are generated. It can be seen in Figure 6.119 that the initial distribution is fairly sparse, and that the trade-off seems to be more or less balanced between the 3 objectives. After 100 generations, and using 20 different weights with binary representation, we can see the Pareto front that contains the best overall result found for this example (see Figure 6.120). The compromise is quite fair between the 3 objectives, with the weight slightly favored. The use of floating point representation produces a very similar Pareto contour, with its best overall result slightly lower than before

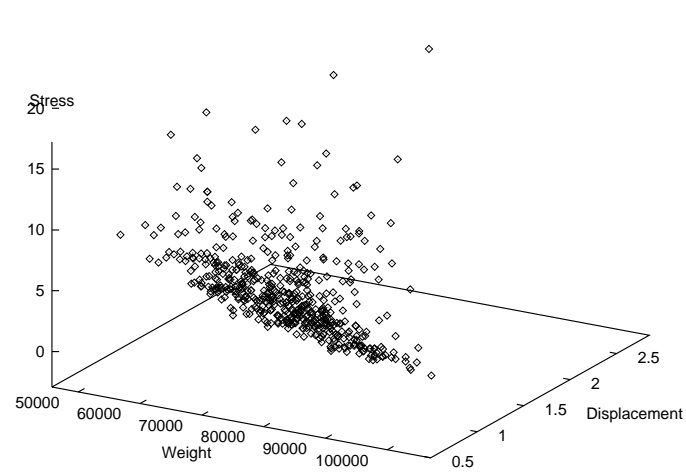


Figure 6.119: Example 4: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero.

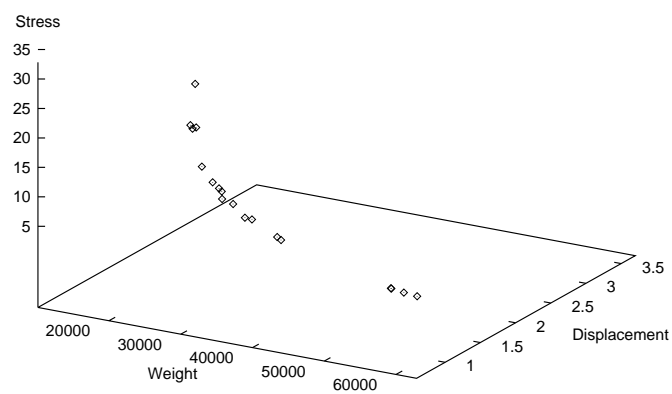


Figure 6.120: Example 4: Distribution of points using my method based on the min-max algorithm with binary representation at generation 100.

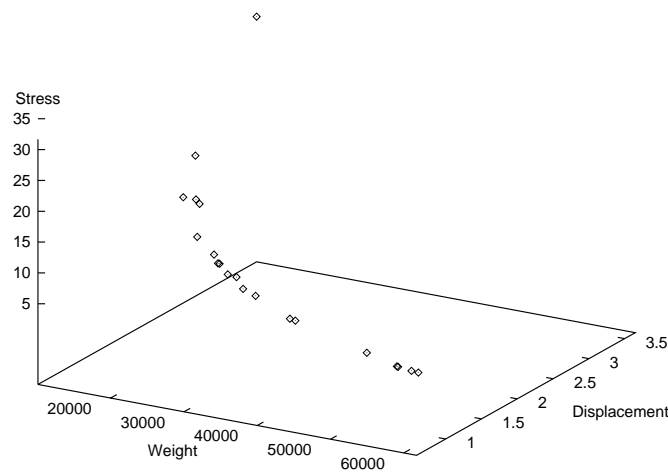


Figure 6.121: Example 4: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 100.

(see Figure 6.121). Apparently the set of weights chosen was very good, because both the Pareto contour and the best overall result were excellent. However, as I have mentioned before, the performance of this technique is highly dependent upon those weights. This is the reason why, it is probably wiser to try to combine the effects of at least two methods when tackling multiobjective optimization problems, so that one can account for the weaknesses of the other.

In this example, I had problems trying to find a good value for  $\sigma_{share}$ . The results presented here correspond to a value of 0.5. Using the same initial population shown in Figure 6.119, we can see that after 20 generations using binary representation, the trend is already clear: displacement and stress have priority over weight. The cluster of points does not look too sparse, even at that early stage of the search, which is an indicator of the premature convergence that we will achieve (see Figure 6.122). One interesting effect produced by this algorithm is that after only 50 generations, it is able to draw the portion of the Pareto front visible from the ranges of the objectives chosen by the GA (see

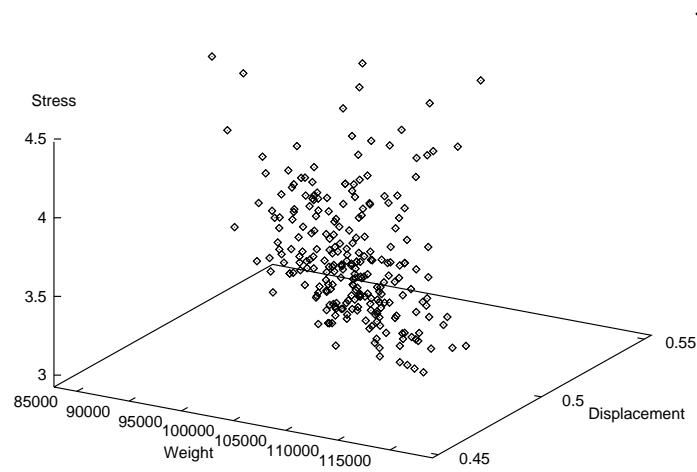


Figure 6.122: Example 4: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20.

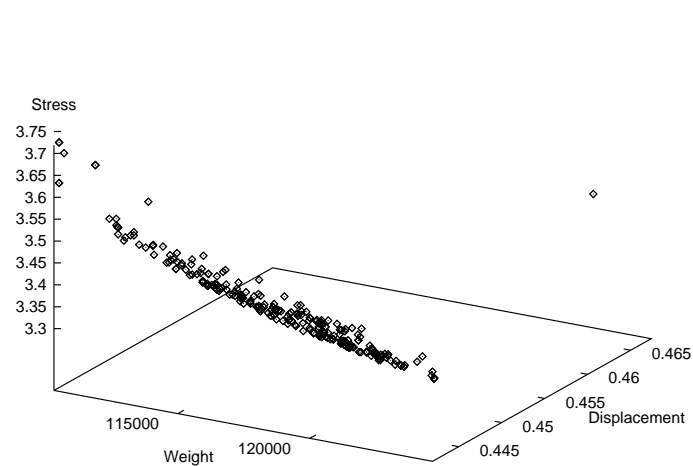


Figure 6.123: Example 4: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 50.

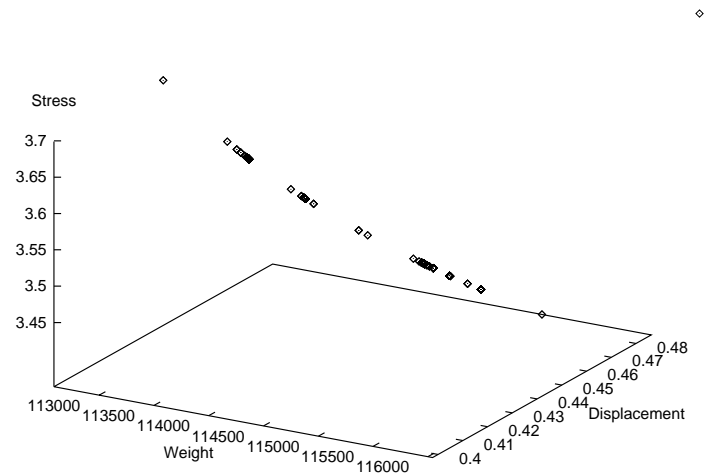


Figure 6.124: Example 4: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100.

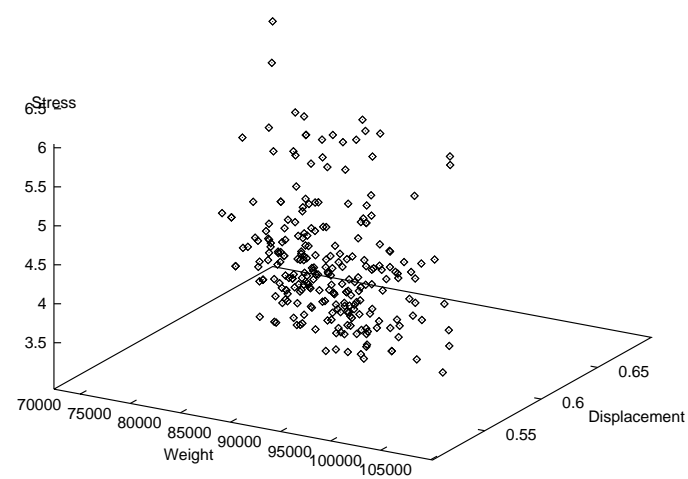


Figure 6.125: Example 4: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20.



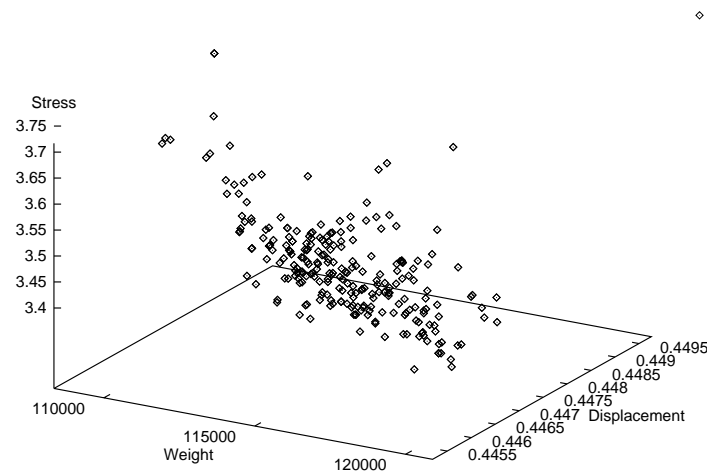


Figure 6.126: Example 4: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 100.

Figure 6.123). The points are lined up in a very rigid manner, with only a couple of dominated results outside the contour. This shows how this technique has the ability to find these Pareto contours relatively easy, but it turns out to be more difficult to keep it from converging to a single global result as we will see in the following graphs. At generation 100 only a few points are visible, delineating the Pareto front, and representing solutions that are optimum in terms of the displacement (the same value of the ideal vector is found) and almost optimum in terms of stress, but not very good in terms of weight (see Figure 6.124). This behavior is typical of the min-max algorithm, since two objectives are optimum or nearly optimum and therefore their deviations are zero. However, in an attempt to achieve such optimality for these two objectives, the method highly disfavor the other one and, consequently, the total deviation is very high, as can be seen in the results (see Tables 6.12, 6.13 and 6.14). After 500 generations, however, the method has converged to a unique solution (in fact, that really happens only a few stages after generation 100). When floating point representation is used, after

20 generations we observe a similar graph as before, which leads us to think that the technique will behave the same way (see Figure 6.125), but that is not the case, because consistency is maintained after 100 generations, without achieving global convergence of the population (see Figure 6.126). The problem now is that the Pareto front does not seem very clear yet. This example shows once more how the representation scheme can make the big difference when using a GA. In this case, even when the solutions found by the two scheme representations are similar (being the best overall solution found using floating point representation better), with trade-offs that favor displacement and stress over weight, the behavior of the technique is more regular (i.e., without premature convergence) when using a representation scheme that normally accelerates convergence.

## 6.6 Example 5 : Design of a 25-bar Space Truss

Once again, the large size of the search space of this problem makes it impractical to show graphically the feasible region. The problem has 3 objective functions, 8 decision variables and 55 constraints. Since the range of each decision variable goes from 0.1 to 999.99, considering increments of 0.01 we would have to perform  $100000^8$  iterations to generate the feasible region (about  $1 \times 10^{40}$  iterations). Considering the amount of time needed to generate all the constraint information and to analyze the structure, this goal seems infeasible in a reasonable amount of time. Nevertheless, as in the previous example, I will provide 3-dimensional representations of the objective function space with each method, to get a good idea of how does the feasible zone look like, and where the Pareto front is. As before, to solve this problem, it was necessary to add a module to each program in order to analyze the space truss generated by each algorithm.

Method	$x_1$	$x_2$	$x_3$	$f_1$	$f_2$	$f_3$
Monte Carlo 1	718.36	24.32	384.19	<b>57144.60</b>	0.050551	1958.00
Monte Carlo 1	716.02	884.66	391.44	275439.48	<b>0.003382</b>	207.27
Monte Carlo 1	701.86	946.54	943.48	232253.56	0.003764	<b>194.88</b>
Min-Max (OS)	3.53	3.53	3.53	<b>1166.98</b>	0.781186	42028.65
Min-Max (OS)	3.88	3.88	3.88	1359.41	<b>0.598842</b>	33872.70
Min-Max (OS)	3.88	3.88	3.88	1359.41	0.598842	<b>33872.70</b>
GA (Binary)	0.100	0.660	3.900	<b>72845.41</b>	1.544286	87294.85
GA (Binary)	808.35	999.99	999.99	330717.40	<b>0.002757</b>	148.303585
GA (Binary)	999.99	999.99	999.99	330717.40	0.002757	<b>148.303585</b>
GA (FP)	0.120	0.220	3.370	<b>468.93</b>	1.565098	90959.54
GA (FP)	999.94	999.99	999.99	330716.80	<b>0.002757</b>	148.303654
GA (FP)	999.99	999.99	999.99	330717.25	0.002757	<b>148.303598</b>
Literature	0.100	0.700	3.200	<b>493.94</b>	1.285167	79916.70
Literature	0.100	0.700	3.200	493.94	<b>1.285167</b>	79916.70
Literature	0.100	0.700	3.200	493.94	1.285167	<b>79916.70</b>

Table 6.16: (Part I) Comparison of results computing the ideal vector of example 5 from Chapter 5 (design of a 25-bar space truss). For each method the best results for optimum  $f_1$ ,  $f_2$  and  $f_3$  are shown in **boldface**. OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.17)

Method	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$
Monte Carlo 1	257.10	739.09	15.86	239.59	11.67
Monte Carlo 1	504.67	731.47	938.00	940.11	984.20
Monte Carlo 1	804.08	943.77	491.77	477.04	765.34
Min-Max (OS)	3.53	3.53	3.53	3.53	3.53
Min-Max (OS)	3.88	3.88	3.88	3.88	5.30
Min-Max (OS)	3.88	3.88	3.88	3.88	5.30
GA (Binary)	0.100	1.29	999.99	0.820	2.560
GA (Binary)	999.99	999.99	999.99	999.99	999.99
GA (Binary)	626.24	885.75	999.99	999.99	999.99
GA (FP)	0.100	3.030	0.750	0.230	3.910
GA (FP)	999.94	999.76	999.79	999.99	999.99
GA (FP)	999.94	999.95	999.99	999.98	999.99
Literature	0.100	1.400	1.100	0.500	3.400
Literature	0.100	1.400	1.100	0.500	3.400
Literature	0.100	1.400	1.100	0.500	3.400

Table 6.17: (Part II) Comparison of results computing the ideal vector of example 5 from Chapter 5 (design of a 25-bar space truss). OS stands for Osyczka's Multiobjective Optimization System.

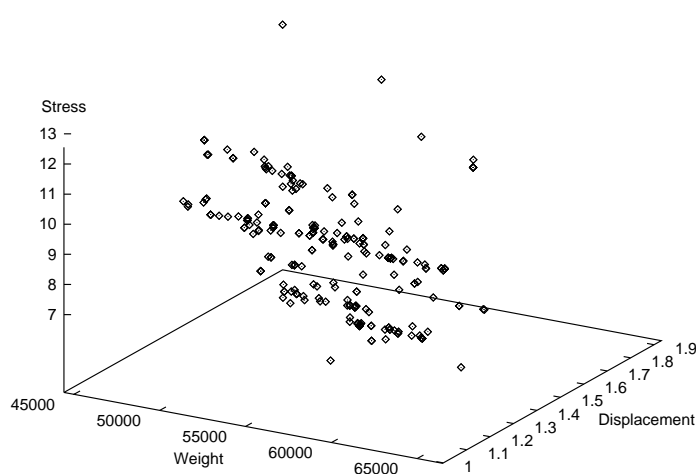


Figure 6.127: Initial feasible region for Monte Carlo method solving the fifth example.

This module uses the matrix factorization method included in Gere and Weaver [202] together with the stiffness method [202] [203] to analyze the structure.

The ideal vector of this problem was computed using Monte Carlo Methods 1 and 2 (generating 300 points) presented in Chapter 4, and a GA (with a population of 300 chromosomes running during 100 generations) using binary and floating point representation, with the procedure described in Chapter 4 to adjust its parameters. The corresponding results are shown in Tables 6.16 and 6.17 including the best results reported in the literature [198].

The results presented for this example demonstrate the inefficiency of mathematical programming techniques to find the ideal vector when the search space is too large. In such circumstances random methods completely fail in finding reasonable solutions and mathematical programming techniques require guessing points too close to the actual solution vector (which we obviously do not know). Again, the GA with floating point representation found the best results, except for the last objective, for which the binary representation found a slightly

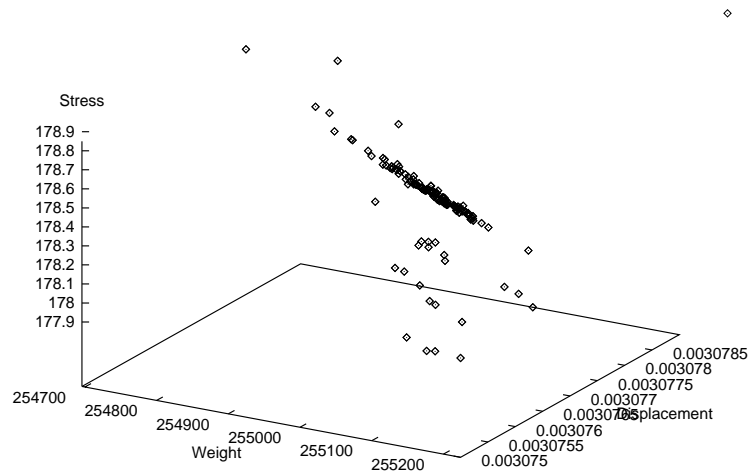


Figure 6.128: Example 5: The GA using a linear combination of the objectives with scaling, after 100 generations using binary representation.

better solution. However, it is interesting that binary representation completely failed to find a reasonable solution for the first objective, probably because of the small population considered for such a long string (136 genes). It is also important to point out that the set of results reported by Coello et al. [198] was produced optimizing only the first objective (i.e., the total weight of the truss) in a discrete manner. As in the previous case, the GA with floating point representation was able to find a better solution than that reported before in the literature. The results for Monte Carlo Method 2 are the same than for Method 1.

Since the Monte Carlo Methods previously mentioned and Osyczka's multiobjective optimization system do not generate the Pareto front, I will compare them with the GA-based approaches only in terms of the best overall result found. Besides those results, it is interesting to observe the initial distribution of points randomly generated by the method (see Figure 6.127). As can be seen from the tables previously shown (see Tables 6.16 and 6.17), the objectives are highly conflicting, and therefore, it is very hard to come up with a good compromise. From

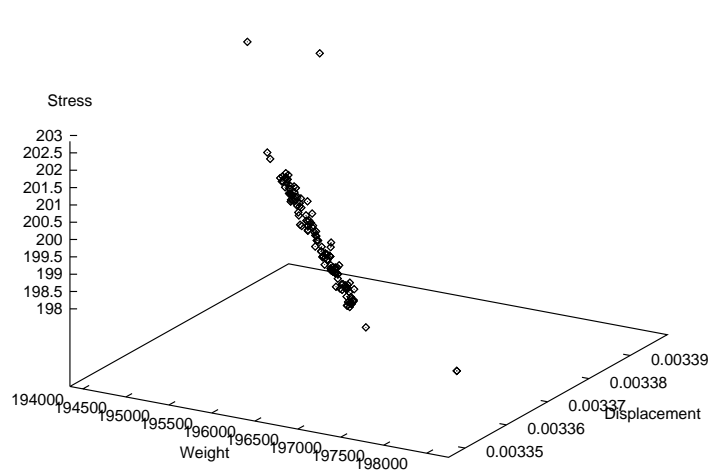


Figure 6.129: Example 5: The GA using a linear combination of the objectives with scaling, after 100 generations using floating point representation.

the graph depicted in Figure 6.127 we can see that the optimum region would be located as close as possible to the ideal vector for each one of the axes. However, because of the great variation in the results, we will see that the common compromise will be to have higher values of weight to allow smaller stresses and displacements. This is true in the physical world, since larger cross-sectional areas allow less stress and displacement in the structure. It is also important observe through all the graphs corresponding to each example the ranges of the axis, because even when the clusters of points could all look alike, there will be a great variation in terms of the ranges.

We will start by analyzing the behavior of the **Lexicographic Method** at different stages of the search process. Figure 6.130 shows the distribution of points produced at generation zero using this method. We can see that this distribution is quite different that the initial distribution of Monte Carlo methods. Here, weight is sacrificed for the sake of getting a lower displacement, but the total stress is very high. In the initial distribution of Monte Carlo methods, points are more

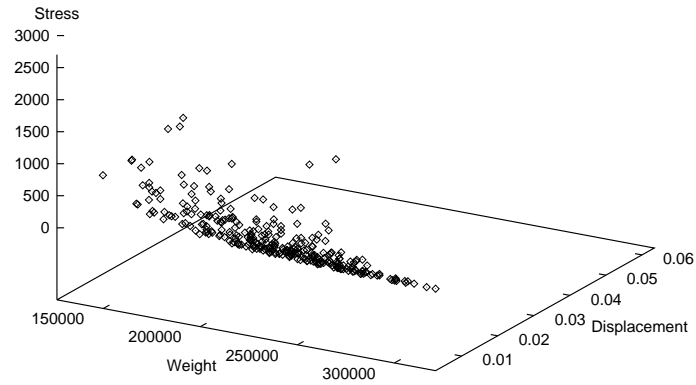


Figure 6.130: Example 5: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed.

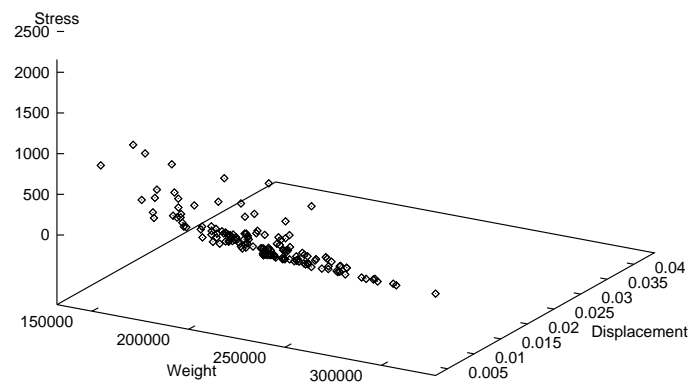


Figure 6.131: Example 5: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed.

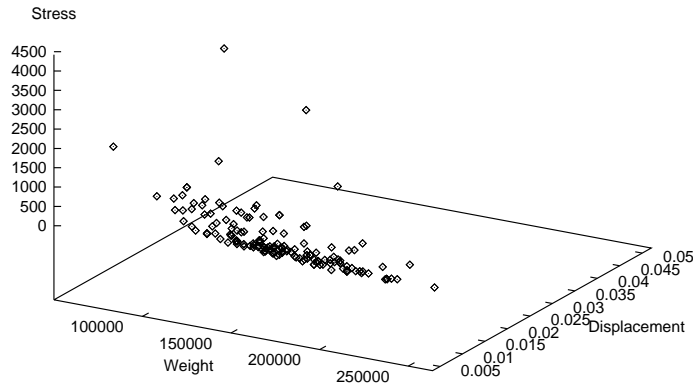


Figure 6.132: Example 5: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed.

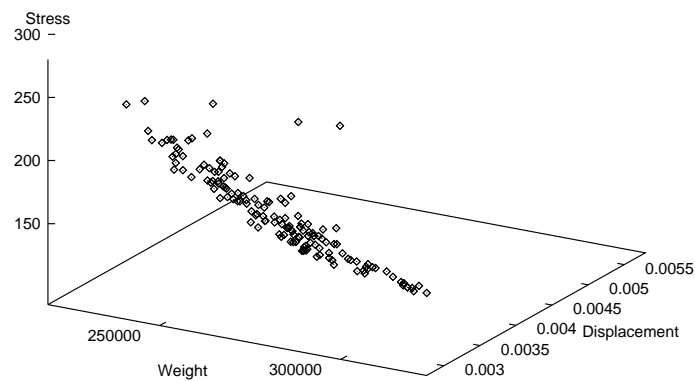


Figure 6.133: Example 5: Distribution of points using the Lexicographic Method with a binary representation at generation 100. Only points within the feasible region are displayed.



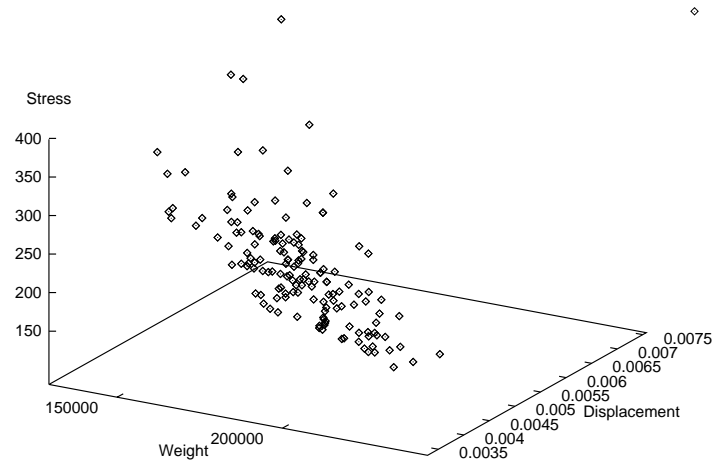


Figure 6.134: Example 5: Distribution of points using the Lexicographic Method with a floating point representation at generation 100. Only points within the feasible region are displayed.

sparsed, and weight has a clear dominance over displacement, but the total stress remains considerably low. This provides a reasonable explanation for the completely different kind of trade-offs that each technique will produce. Figure 6.131 shows the distribution of points at generation 20 using binary representation, and Figure 6.132 shows the corresponding distribution using floating point representation. The trade-offs provided by both graphs are different, because binary representation is favoring stress over weight, whereas floating point representation favors weight over stress. Displacement, on the other hand, has similar values in both graphs. In this example, contrary to the results of the previous one, binary representation produces a more uniform distribution of points that floating point representation. After 100 generations, binary representation definitely shows a better distribution of points, almost conforming a plane that seems to be the Pareto front (see Figure 6.134). Floating point representation, shows a cluster of points much more sparsed (see Figure 6.134). However, in terms of the best overall result, floating point representation can do much better, since it keeps a better

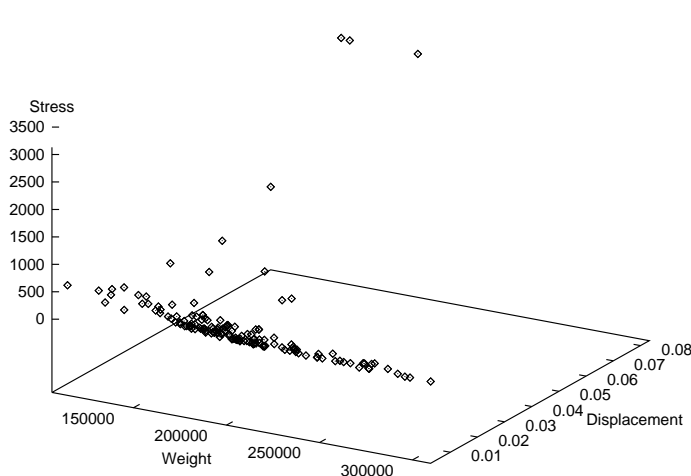


Figure 6.135: Example 5: Distribution of points using VEGA with a binary representation at generation twenty.

trade-off among the objectives, with a better value for the weight than when using binary representation, which favors a higher weight to reduce the total stress. As before, I also compared the effect of a simple linear combination of objectives (addition or multiplication) using scaling. A linear combination of objectives did not produce good results in this example. When a binary representation was used, the population lined up into a straight line after 100 generations, with only a few points outside it (see Figure 6.128). Unfortunately this line represents trade-offs that are not very well balanced, since the values of the three objectives are quite high. The use of floating point representation improved things a little bit, producing lower weights while keeping the same stress as before (i.e., when using binary representation) and allowing a higher stress (see Figure 6.129). This turns out to produce better overall solutions, and a more reasonable contour of the Pareto front. However, the trade-offs are still too high and the arrangement of the points at generation 100 are a clear indicator of an imminent convergence of the entire population, which happens only a few generations later.

Method	$x_1$	$f_1$	$f_2$	$f_3$	$L_p(f)$
Ideal Vector		468.928261	0.002757	148.303585	0.000000
Monte Carlo 1	786.01	113293.85	0.006212	363.6076	243.306621
Monte Carlo 2	660.75	110264.89	0.006925	394.8020	237.316252
Min-max (OS)	3.8821	1344.32	0.676830	34793.19	479.969778
GCM (OS)	3.8821	1359.41	0.598842	33872.70	445.507894
WMM (OS)	3.8821	1344.32	0.676830	34793.19	479.969778
PMM (OS)	3.8821	1359.41	0.598842	33872.70	445.50789
NMM (OS)	3.8821	1359.41	0.598842	33872.70	445.50789
GALC (B)	282.44	254696.03	0.003078	178.85	542.46737
GALC (FP)	247.91	193849.17	0.003389	202.83	412.98462
Lexicographic (B)	516.62	219176.78	0.005791	280.06	468.38825
Lexicographic (FP)	450.57	129424.79	0.005412	303.46	277.010455
VEGA (B)	839.62	234854.32	0.003473	205.70	500.478800
VEGA (FP)	468.34	219453.21	0.003482	202.59	467.617915
NSGA (B)	999.99	250615.78	0.002975	171.74	533.680836
NSGA (FP)	709.53	226478.31	0.003971	205.53	482.796241
MOGA (B)	83.50	85297.74	0.023970	990.87	194.274927
MOGA (FP)	201.80	81778.41	0.021254	908.03	185.226194
NPGA (B)	28.41	92943.08	0.010969	585.22	203.127877
NPGA (FP)	539.39	55812.18	0.029665	1307.44	135.596546
Hajela (B)	6.88	99464.52	0.017495	1134.42	223.105294
Hajela (FP)	69.95	107947.60	0.007593	421.48	232.796738
GAminmax1 (B)	9.28	85604.05	0.036615	2190.83	207.605910
GAminmax1 (FP)	8.42	16230.99	0.037474	2227.73	60.226711
GAminmax2 (B)	999.99	330717.40	0.002757	148.30	704.262255
GAminmax2 (FP)	921.03	322935.45	0.002769	149.59	687.680069

Table 6.18: (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the fifth example (design of a 25-bar space truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Table 6.19)

Method	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$
Monte Carlo 1	441.86	412.95	221.47	593.79	240.96	37.1945	642.65
Monte Carlo 2	291.60	478.25	629.48	777.60	187.36	56.9828	577.80
Min-max (OS)	3.8821	5.2964	3.8821	3.8821	3.8821	3.8821	3.8821
GCM (OS)	3.8821	3.8821	3.8821	3.8821	3.8821	3.8821	5.2964
WMM (OS)	3.8821	5.2964	3.8821	3.8821	3.8821	3.8821	3.8821
PMM (OS)	3.8821	3.8821	3.8821	3.8821	3.8821	3.8821	5.2964
NMM (OS)	3.8821	3.8821	3.8821	3.8821	3.8821	3.8821	5.2964
GALC (B)	425.67	999.99	59.00	123.80	999.99	820.83	999.99
GALC (FP)	440.14	999.39	0.52	513.61	439.53	493.92	998.86
Lexicographic (B)	455.59	587.54	999.99	932.43	999.99	761.67	999.99
Lexicographic (FP)	439.90	520.77	427.54	296.86	375.58	725.23	655.95
VEGA (B)	999.99	999.99	999.99	661.79	853.86	999.99	999.99
VEGA (FP)	920.78	870.16	418.78	839.55	817.89	947.87	985.55
NSGA (B)	999.99	999.99	599.88	999.99	977.42	999.99	999.99
NSGA (FP)	887.07	982.84	982.05	780.48	890.77	761.94	906.92
MOGA (B)	344.84	799.85	351.95	968.68	447.63	102.92	82.12
MOGA (FP)	766.06	290.23	795.97	539.97	895.71	399.41	351.92
NPGA (B)	652.02	740.52	761.61	966.87	464.42	635.52	575.39
NPGA (FP)	177.03	175.81	889.32	514.16	103.14	2.91	117.61
Hajela (B)	40.84	171.33	1.27	147.24	999.99	10.41	277.95
Hajela (FP)	569.04	416.87	642.84	250.29	112.66	85.28	743.15
GAminmax1 (B)	11.66	88.95	3.02	77.09	999.99	3.95	127.39
GAminmax1 (FP)	14.53	104.01	0.14	77.21	43.19	4.91	117.57
GAminmax2 (B)	999.99	999.99	999.99	999.99	999.99	999.99	999.99
GAminmax2 (FP)	997.60	999.59	961.44	988.96	998.70	989.70	998.41

Table 6.19: (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the fifth example (design of a 25-bar space truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective).

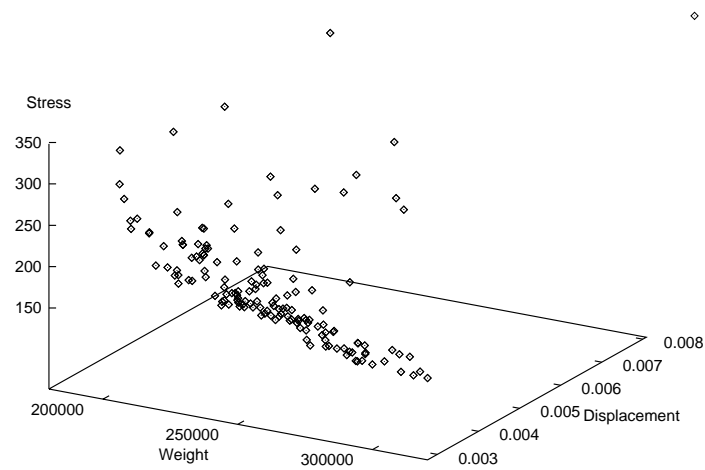


Figure 6.136: Example 5: Distribution of points using VEGA with a binary representation at generation fifty.

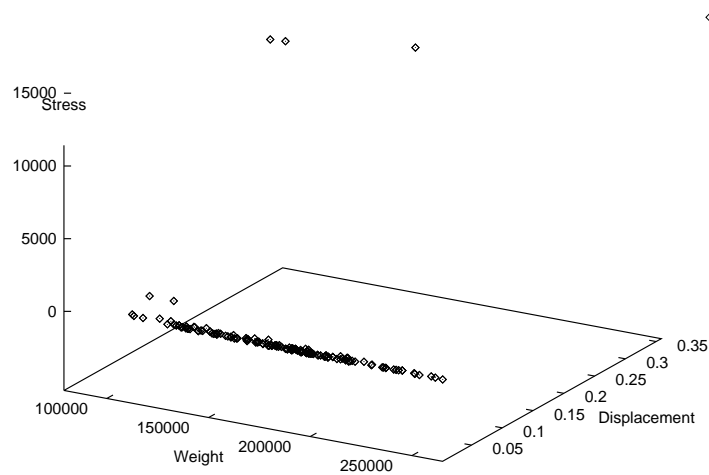


Figure 6.137: Example 5: Distribution of points using VEGA with floating point representation at generation twenty.

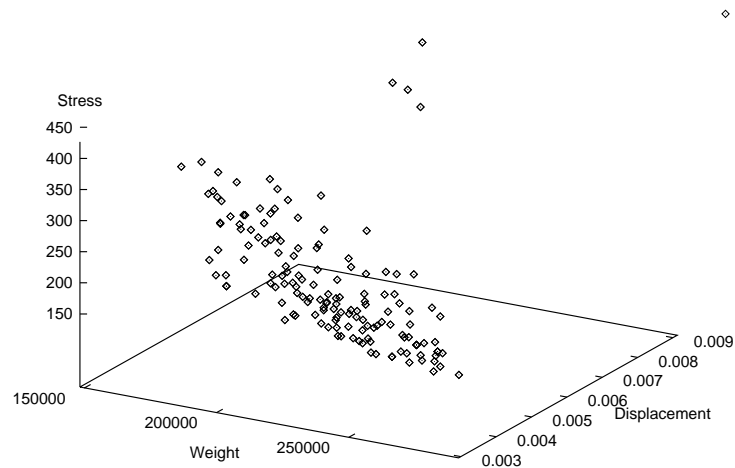


Figure 6.138: Example 5: Distribution of points using VEGA with floating point representation at generation fifty.

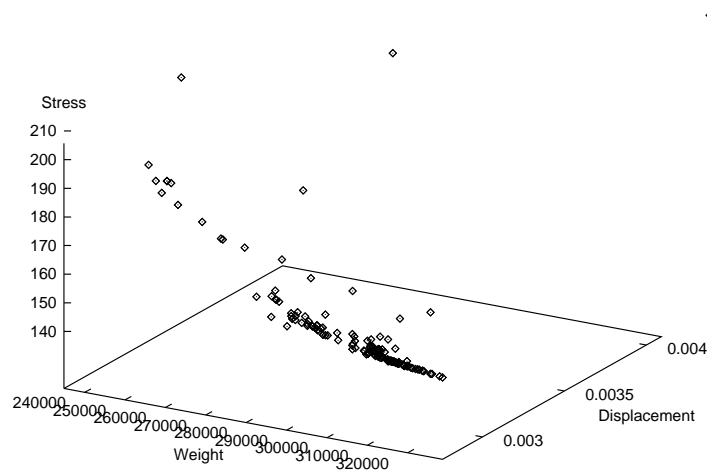


Figure 6.139: Example 5: Distribution of points using VEGA with binary representation at generation 100.

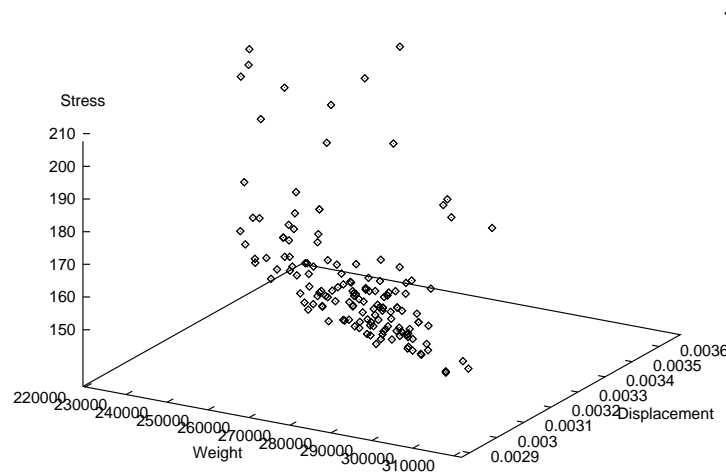


Figure 6.140: Example 5: Distribution of points using VEGA with floating point representation at generation 100.

Schaffer's VEGA produces a very compact distribution of points after 20 generations using binary representation (see Figure 6.135), with only a few dominated points within the population. Weight has a clear priority over stress and displacement up to this point. After 50 generations, the population looks more sparse, and the priorities have changed, giving stress and displacement preference over weight (see Figure 6.136). After 100 generations, we have two contours, from which the most compact seems to be the Pareto front (see Figure 6.139). Displacement and stress have decreased and, consequently, weight has increased. The best overall solution produced using this method with binary representation is extremely poor, because even when stress and displacement are extremely close to the optimum values, weight is far too high to produce a reasonably good maximum deviation. The results are very similar using floating point representation, because after only 20 generations the graph presents the form of a straight line of points that represent compromises in which stress is extremely high and

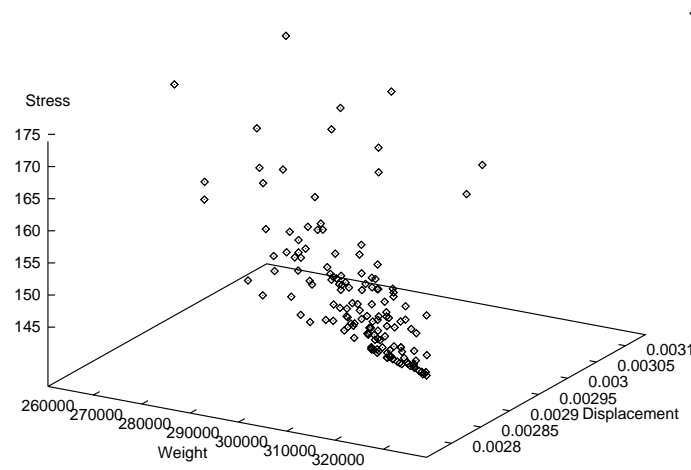


Figure 6.141: Example 5: Distribution of points using NSGA with binary representation at generation 100.

displacement and weight are conservatively low (see Figure 6.137). After 50 generations, however, weight starts increasing, and stress and displacement go down, producing better trade-offs (see Figure 6.138). Finally, after 100 generations, the population is still very sparse, but the trend is clear: displacement and stress are converging towards their optimum values, whereas the weight is increased (see Figure 6.140). Due to the less uniform distribution of points at this generation, I was able to find a better overall result using floating point representation. I should also mention the fact that, as it will be seen in the following graphs, a situation that kept repeating in this problem was that floating point representation was able to keep more diversity in the population than binary representation, which is not a common phenomenon, since this representation normally works all the other way around.

After 20 generations, my version of Srinivas' NSGA that uses binary representation presents a quite sparse distribution of solutions in which displacement



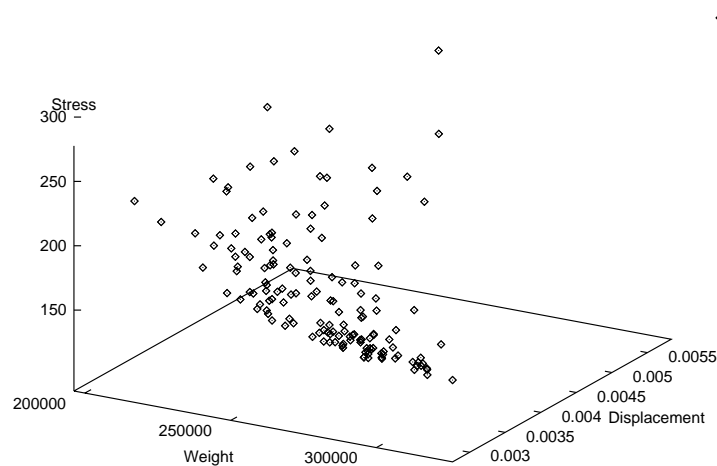


Figure 6.142: Example 5: Distribution of points using NSGA with binary representation at generation twenty.

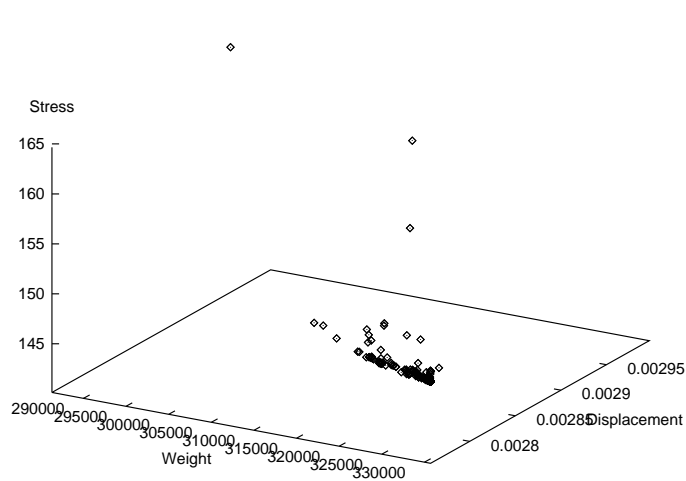


Figure 6.143: Example 5: Distribution of points using NSGA with binary representation at generation 500.

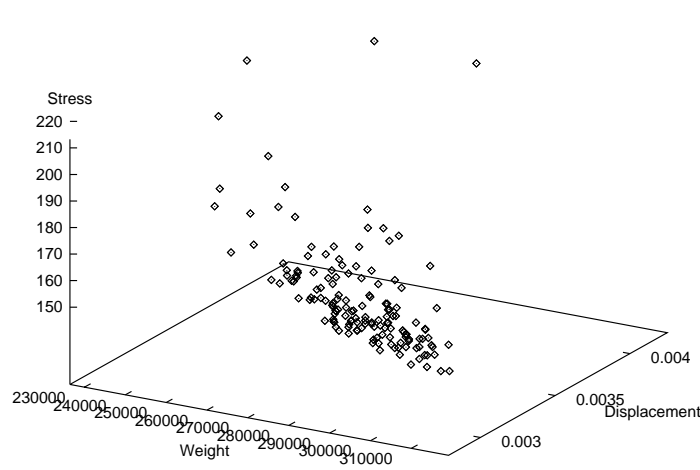


Figure 6.144: Example 5: Distribution of points using NSGA with floating point representation at generation 100.

and stress are strongly favored over weight (see Figure 6.142). After 100 generations, this trend is even stronger, producing in consequence higher weights and lower stresses (see Figure 6.141). The points are still very sparsely distributed, although there is an important grouping in a zone where weight is very high and stress and displacement are practically optimum. After 500 generations, the population has almost converged to a point within the zone previously mentioned, in which weight is extremely high, but stress and displacement are practically optimum (see Figure 6.143). Again, the selection mechanism of this technique seems to favor solutions that highly dominate with respect to those two objectives, sacrificing compromises that would balance better with respect to weight. The use of floating point representation after 100 generations produces similar results as before (i.e., with binary representation), but the weight is kept slightly lower (see Figure 6.144). Because of that reason the best overall solution is better when floating point representation is used. Once again, there was no significant change in the results when the value of  $\sigma_{share}$  was modified.

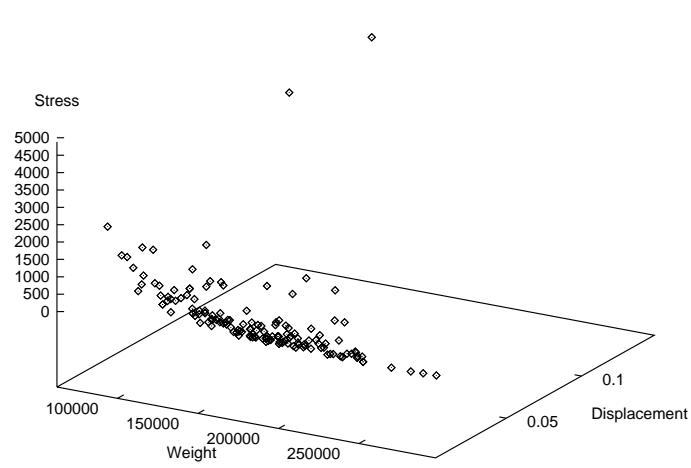


Figure 6.145: Example 5: Distribution of points using MOGA with binary representation at generation 20.

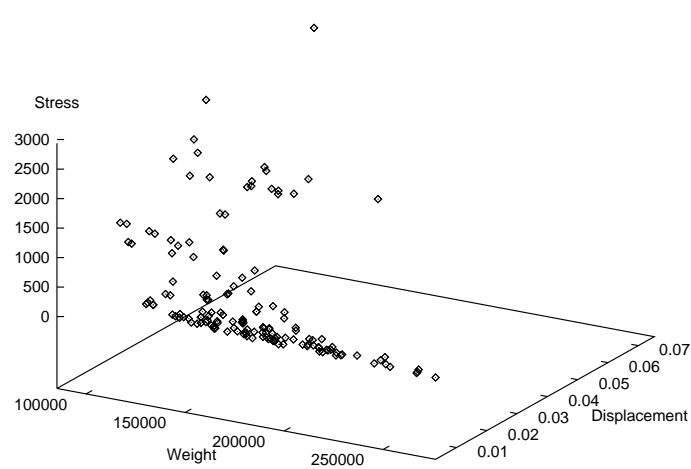


Figure 6.146: Example 5: Distribution of points using MOGA with binary representation at generation 100.

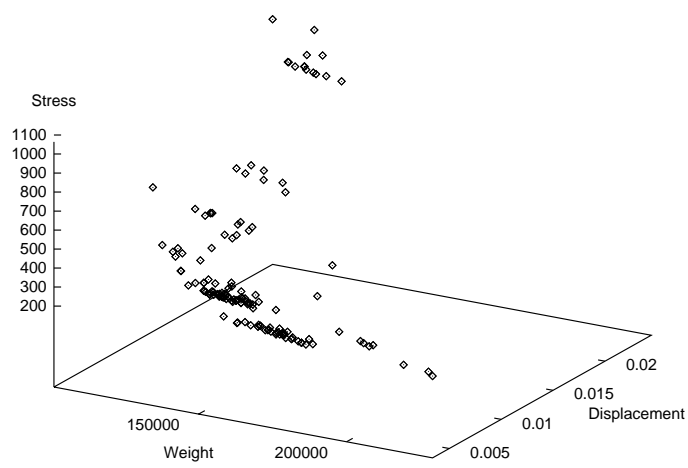


Figure 6.147: Example 5: Distribution of points using MOGA with binary representation at generation 500.

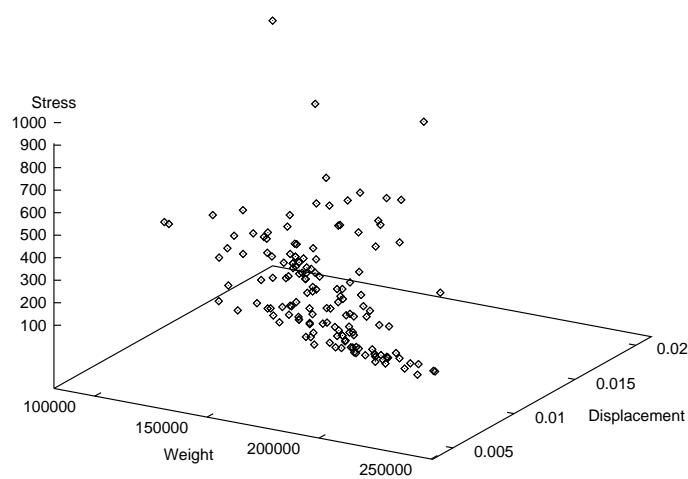


Figure 6.148: Example 5: Distribution of points using MOGA with floating point representation at generation 100.

Fonseca's MOGA gives a good region of compromise solutions after 20 generations using binary representation, favoring weight over displacement and stress (see Figure 6.145). The Pareto front seems very clear in this case, with only a few dominated points. After 100 generations, the ranges of stress and displacement have been lowered, but without increasing weight, showing the right way to follow in terms of finding the best trade-off. Most of the Pareto front is still there, but there are more dominated points within the population (see Figure 6.146). Due to the good compromise solutions present at this stage of the search, the best overall result reported at this point is remarkably good. After 500 generations, the method shows a set of points less compact than at earlier stages in the search, with better compromise solutions (i.e., solutions that favor the three objectives), but without clear indication of where is the Pareto front (see Figure 6.147). The best overall solution found by this technique is excellent using both representations, being the one produced using floating point representation the best found so far. The graph after 100 generations using floating point representation shows more sparse points than when using binary representation, but the compromises are better since the ranges of the objectives are slightly smaller for the three objectives (see Figure 6.148). I also used a value of  $\sigma_{share} = 0.1$  as in NSGA to generate these graphs, but changing this value did not affect the results produced by the algorithm.

After 20 generations, NPGA produces a sparse distribution of points, with trade-offs that seem very reasonable, since none of the objectives has excessively high values (see Figure 6.149). After 50 generations, things have not changed much, and we only observe that some points start to gather at a region parallel to the plane formed by weight and displacement (see Figure 6.150). At generation 100, the trend seems clear: weight is being favored over displacement and

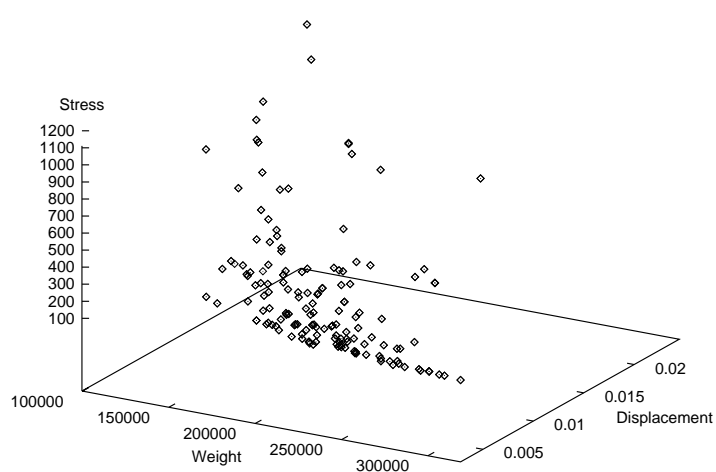


Figure 6.149: Example 5: Distribution of points using NPGA with binary representation at generation 20.

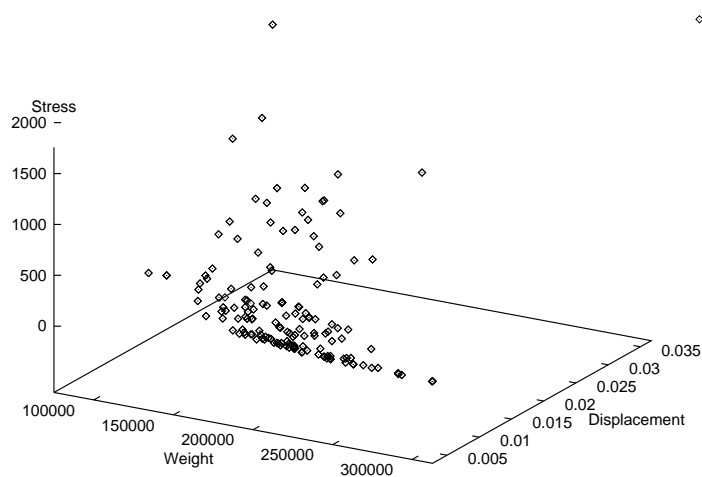


Figure 6.150: Example 5: Distribution of points using NPGA with binary representation at generation 50.

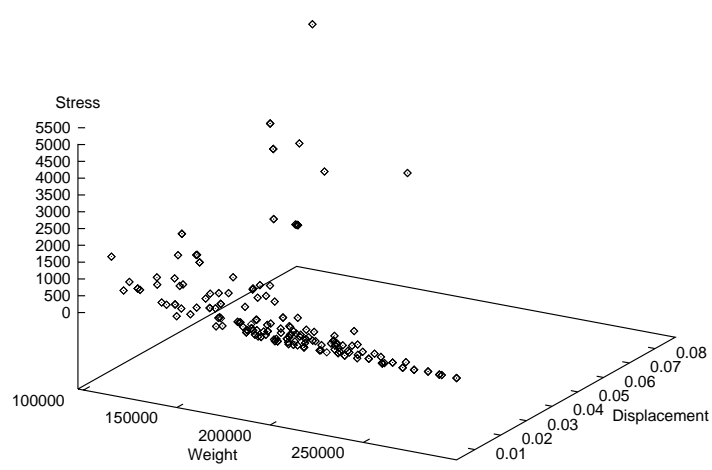


Figure 6.151: Example 5: Distribution of points using NPGA with binary representation at generation 100.

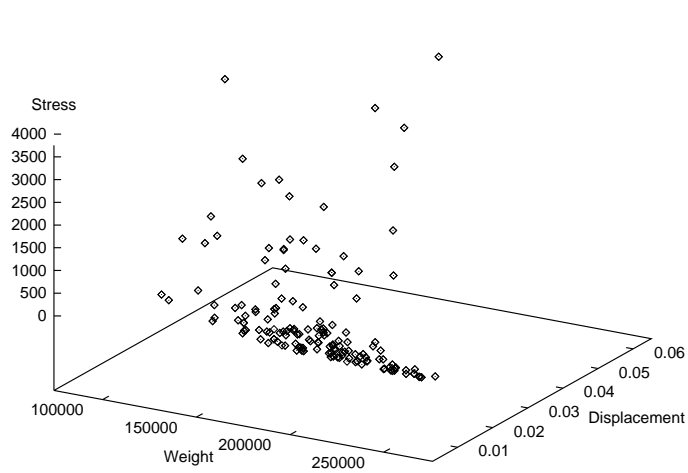


Figure 6.152: Example 5: Distribution of points using NPGA with binary representation at generation 500.

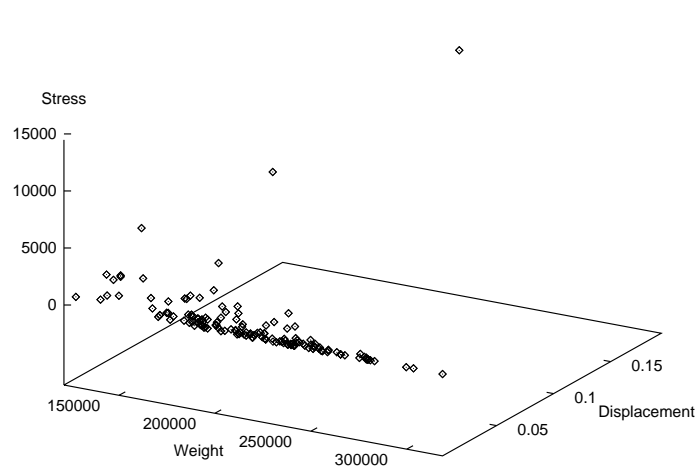


Figure 6.153: Example 5: Distribution of points using NPGA with binary representation at generation 100 with  $\sigma_{share} = 1.0$ .

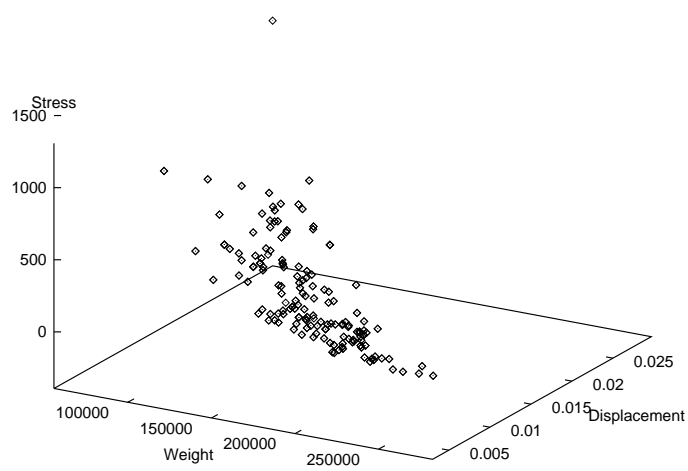


Figure 6.154: Example 5: Distribution of points using NPGA with floating point representation at generation 100.



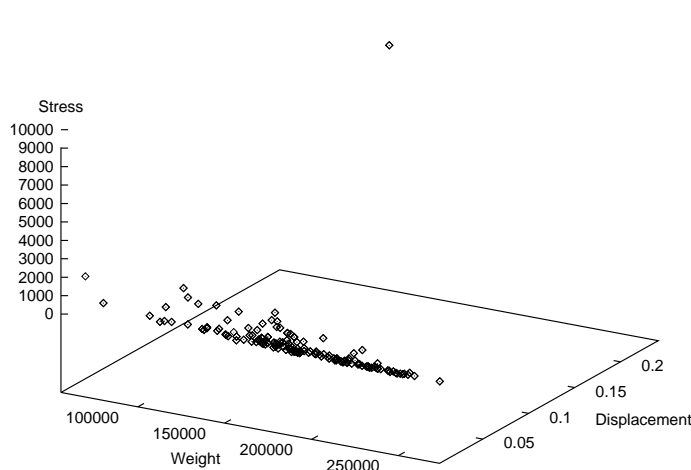


Figure 6.155: Example 5: Distribution of points using NPGA with floating point representation at generation 20.

stress (see Figure 6.151), but good trade-offs are being made, since the values of stress and displacement are not excessive. This is corroborated by the best overall solution reported (see Tables 6.16 and 6.17). The main strength of this method is proved once more, since after 500 generations the diversity of the population is maintained, with a very solid region parallel to the planes formed by the weight and displacement, slightly more sparse than at generation 100 (see Figure 6.152). As in previous examples, this method was able to keep more points through many generations than any other Pareto-based technique. The use of floating point representation produces a similar effect at generation 20, only that this time the population forms a more compact cluster of points shaped as a straight line, defining the Pareto front (see Figure 6.155). After 100 generations, the Pareto front starts to vanish, but the compromises remain the same, with good values for every objective (see Figure 6.154). The sparse distribution of the population at this stage of the search must be the reason why floating point representation produced a better overall solution than binary representation. As in

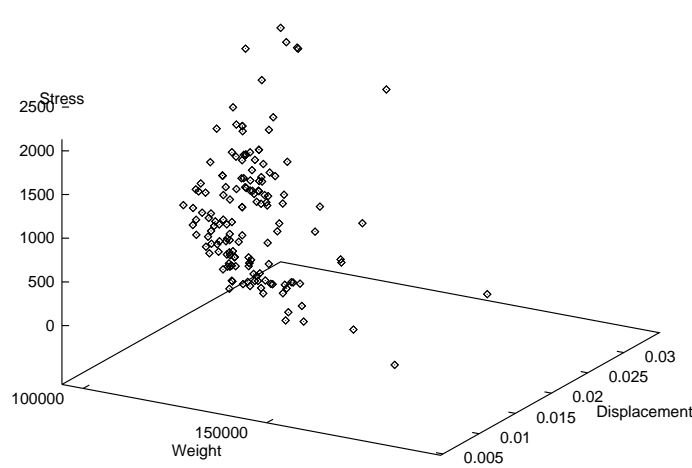


Figure 6.156: Example 5: Distribution of points using Hajela's method with binary representation at generation 20.

previous examples, the value of  $\sigma_{share}$  plays a critical role in the performance of the algorithm, because if we increase it from 0.1 (same value used with the previous techniques that use niches) to 1.0, the distribution of points is less compact than before, slightly favoring displacement over weight (see Figure 6.153).

Hajela's method provides moderately good results as in the previous example, but it can not keep them through too many generations. After 20 generations using binary representation, the distribution of points is sparse, but a sector of the Pareto front is already visible, with compromises in which weight and displacement take priority over stress (see Figure 6.156). After 50 generations the trade-off remains the same, but now the weight has gotten slightly lower, allowing to display a clear image of the Pareto front (see Figure 6.157). At generation 100, the Pareto front gets thinner, with a lowering of stresses and an increment of weight (see Figure 6.158). Total convergence to a unique solution seems only a matter of time, and after 500 generations only 3 points remain in the graph, as expected (see Figure 6.159). The best overall results found by this technique using both

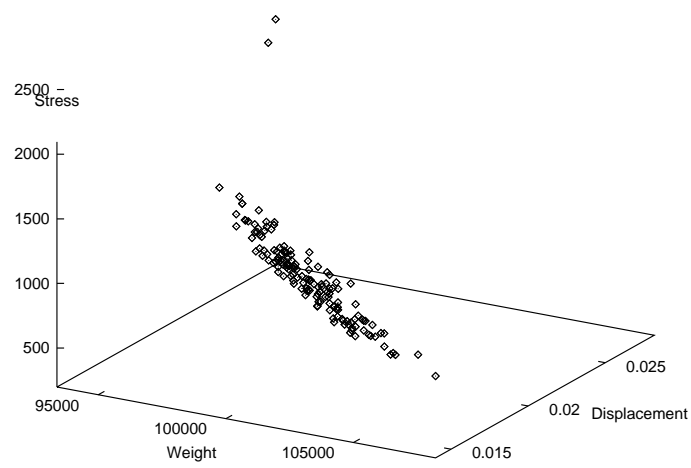


Figure 6.157: Example 5: Distribution of points using Hajela's method with binary representation at generation 50.

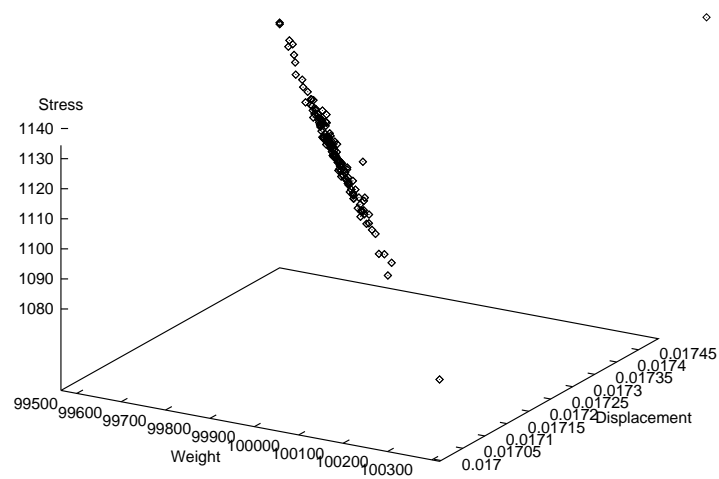


Figure 6.158: Example 5: Distribution of points using Hajela's method with binary representation at generation 100.

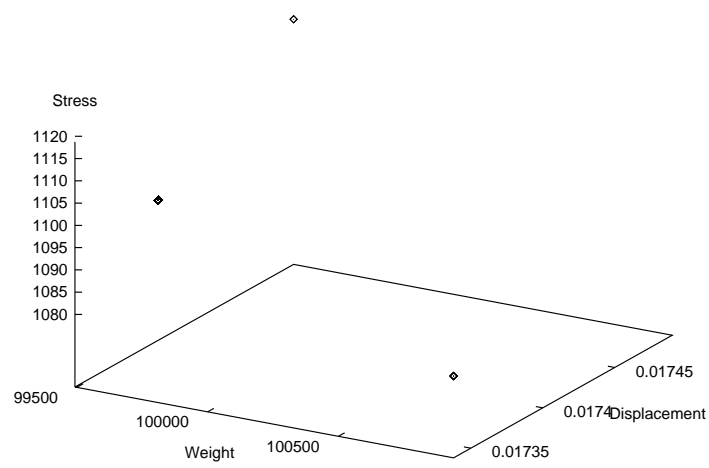


Figure 6.159: Example 5: Distribution of points using Hajela's method with binary representation at generation 500

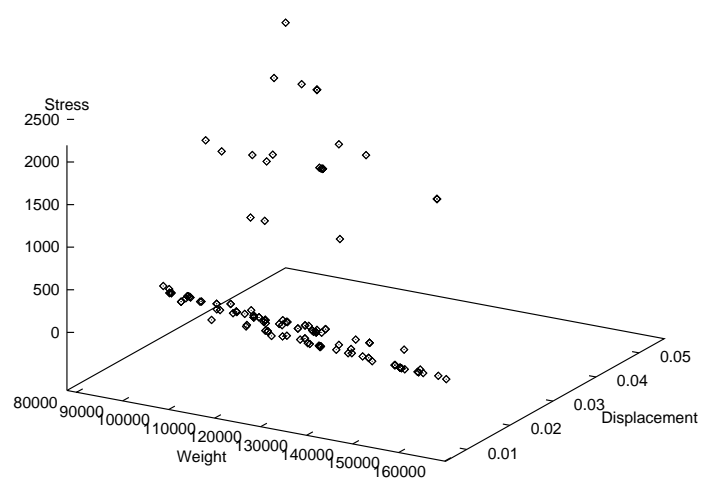


Figure 6.160: Example 5: Distribution of points using Hajela's method with floating point representation at generation 20.

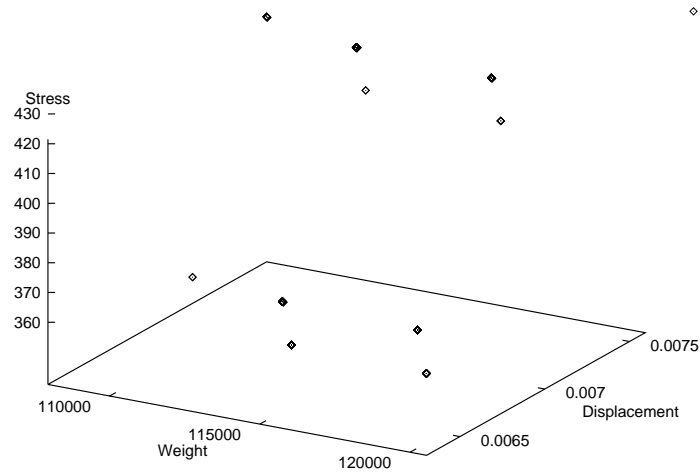


Figure 6.161: Example 5: Distribution of points using Hajela's method with floating point representation at generation 100.

representations are above average, mainly because the value of the third objective is kept low. Binary representation provides, in this case, a slightly better overall result. When using floating point representation, the population at generation 20 seems to be forming a solid front that balances the three objectives quite well (see Figure 6.160). However, the method is not able to keep such front, and after 100 generations the technique has converged to a few points none of which represent better overall solutions than those found using binary representation (see Figure 6.161). The change of representation produced the same premature convergence as before, only that faster, because of the accelerated convergence property of this representation scheme.

Before showing the results produced with my method based on the min-max approach, I want to show the distribution of points at generation zero, since in this case the algorithm ensures that only feasible solutions are generated. It can be seen in Figure 6.162 that the initial distribution looks already very well distributed along what seems to be the Pareto front. Apparently the high number

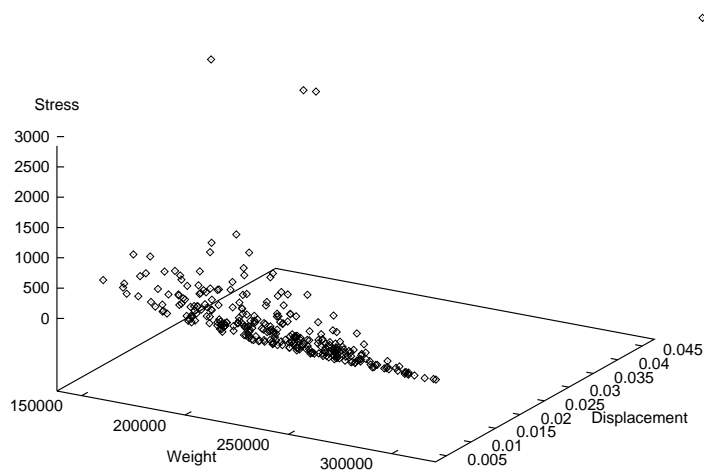


Figure 6.162: Example 5: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero.

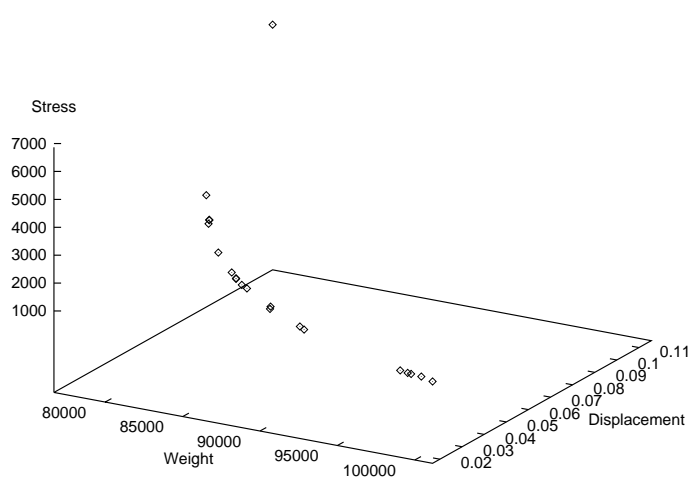


Figure 6.163: Example 5: Distribution of points using my method based on the min-max algorithm with binary representation at generation 100.

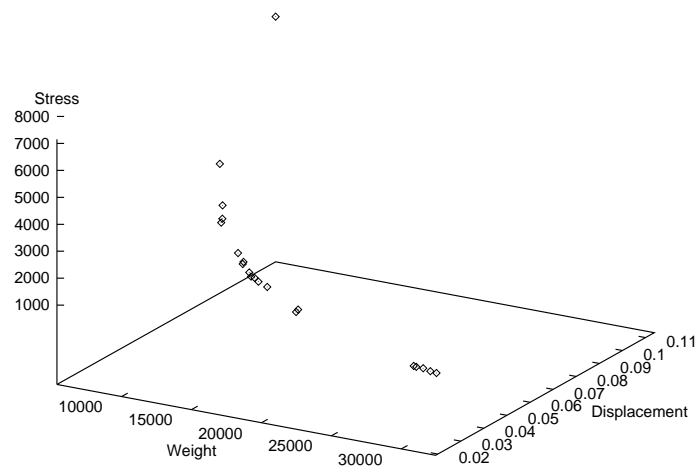


Figure 6.164: Example 5: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 100.

of constraints seems to be an engine good enough to drive the search towards the proper zone. The trade-off is also very good, with low displacements and weights, and relatively low weights. After 100 generation and using 20 different weights with binary representation, we can see the Pareto front very clearly (see Figure 6.163). Notice how weight and displacement are kept low whereas stress is relatively high. The amount of points displayed, even when small, seems sufficient to sketch the Pareto contour. If we use floating point representation, then the contour will look similar after 100 generations (see Figure 6.164), but looking at the ranges of the objectives we will notice how the weight is now lower whereas the stress is kept in about the same range as before (i.e., when using binary representation). This representation scheme provided, by far, the best overall solution that I was able to get for this problem, with an extremely low deviation. This corroborates, once more, my hypothesis about the efficiency of this method both for computing the best overall result and for getting the Pareto front, as long as we select an appropriate set of weights. The results obtained in this and

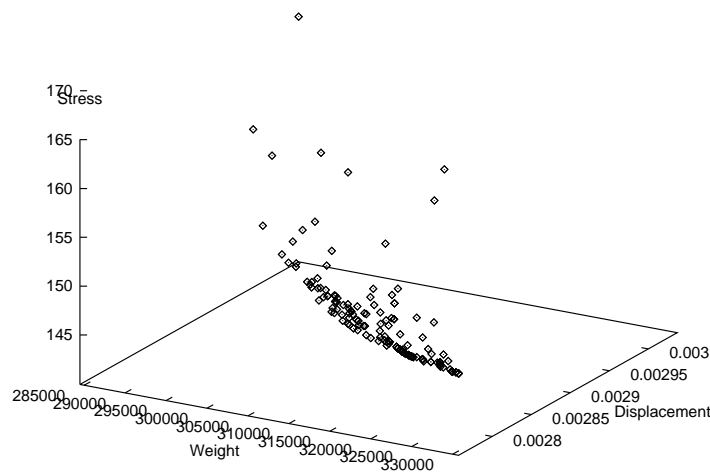


Figure 6.165: Example 5: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20.

previous examples shows how this technique can be an excellent substitute for any mathematical programming technique, since it has systematically provided a much better overall result in every single problem in which it has been used.

As in the previous example, I also had a lot of problems trying to find a good value for  $\sigma_{share}$ . The results presented here correspond to a value of 0.5. Using the same initial population shown in Figure 6.162, we can see that after 20 generations using binary representation, the trend is already clear: displacement and stress have priority over weight. The cluster of points does not look too sparse, even at that early stage of the search, which is an indicator of the premature convergence that we will achieve (see Figure 6.165). After 50 generations the entire population has converged to a unique solution in which displacement and stress are optimum and the weight is extremely high. The use of floating point representation provides a better perspective at generation 20, since there is still a sparse distribution of points representing solutions in which weight is lower than when using binary representation, and displacement and stress are kept



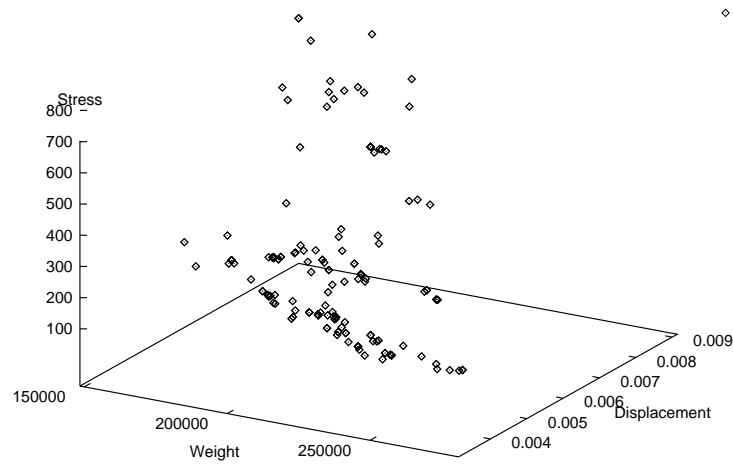


Figure 6.166: Example 5: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20.

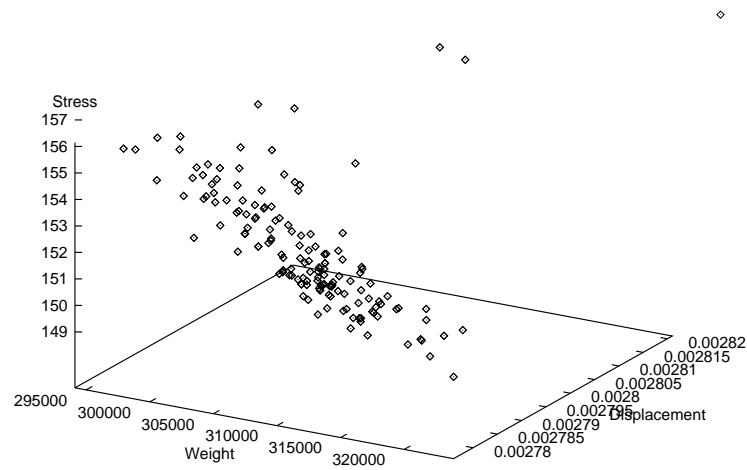


Figure 6.167: Example 5: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 50.

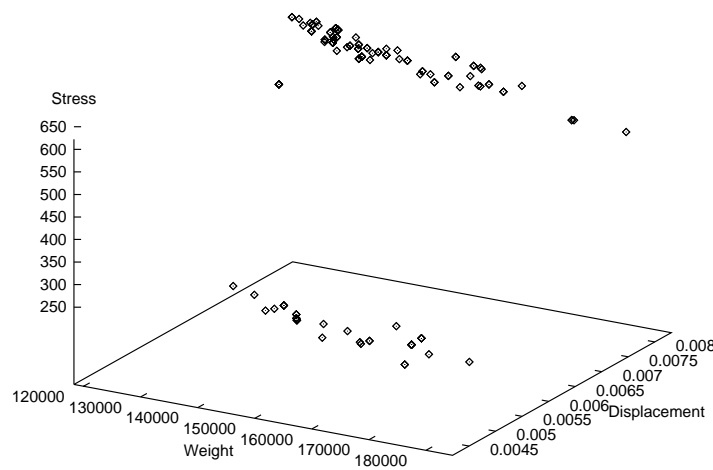


Figure 6.168: Example 5: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 100.

relatively low (see Figure 6.166). At generation 50, weight has increased and several points have achieved optimality with respect to displacement and stress. However, the population still looks fairly sparse (see Figure 6.167). At generation 100 we have almost total convergence of the population, although the values of the displacements are higher than before (when using binary representation) and the values of the weights are lower (see Figure 6.168). The premature convergence of the algorithm shows the main drawback of this method: it converges too quickly when more than one objective may achieve optimality at the same time, even if the remaining objective (or objectives) have extremely high values (see Tables 6.18 and 6.19). Floating point representation provided a slightly better best overall solution, but in general we may say that their solutions after 100 generations (when total convergence has practically been achieved) are quite similar.

## 6.7 Example 6 : Design of a 200-bar Plane Truss

Method	$x_1$	$x_2$	$x_3$	$f_1$	$f_2$	$f_3$
Monte Carlo 1	488.72	110.45	546.89	<b>3019191.78</b>	3.9983	74.99
Monte Carlo 1	569.46	29.17	760.92	5423359.65	<b>0.522427</b>	9.5056
Monte Carlo 1	836.72	135.62	41.77	5635985.90	0.556857	<b>8.7115</b>
Min-Max (OS)	47.418	47.418	47.418	<b>617227.10</b>	3.8398	91.9799
Min-Max (OS)	49.601	49.601	49.601	650984.03	<b>3.6148</b>	86.0389
Min-Max (OS)	49.601	49.601	49.601	641862.42	3.6235	<b>85.8669</b>
GA (Binary)	0.300	1.290	0.100	<b>893885.79</b>	33.6943	957.1493
GA (Binary)	999.99	999.99	999.99	9963295.72	<b>0.370375</b>	5.124250
GA (Binary)	999.99	999.99	896.22	9963295.72	0.370375	<b>5.124250</b>
GA (FP)	0.100	1.140	0.100	<b>36167.73</b>	38.675848	1062.77608
GA (FP)	999.58	999.50	995.30	9961698.96	<b>0.370376</b>	5.124382
GA (FP)	999.21	999.78	997.52	9962313.62	0.370377	<b>5.124336</b>
Literature	0.100	1.504	1.202	<b>35162.93</b>	44.144661	1137.7476
Literature	0.100	1.504	1.202	35162.93	<b>44.144661</b>	1137.7476
Literature	0.100	1.504	1.202	35162.93	44.144661	<b>1137.7476</b>

Table 6.20: (Part I) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). For each method the best results for optimum  $f_1$ ,  $f_2$  and  $f_3$  are shown in **boldface**. OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.21, 6.22, 6.23 and 6.24)

Method	x <sub>4</sub>	x <sub>5</sub>	x <sub>6</sub>	x <sub>7</sub>	x <sub>8</sub>	x <sub>9</sub>	x <sub>10</sub>
Monte Carlo 1	863.54	725.49	86.99	370.21	320.60	342.01	39.47
Monte Carlo 1	205.42	798.62	793.28	26.42	88.80	42.19	330.31
Monte Carlo 1	285.47	280.01	300.94	227.05	218.25	858.61	722.51
Min-Max (OS)	47.418	47.418	47.418	47.418	47.418	47.418	47.418
Min-Max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
Min-Max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GA (Binary)	0.100	5.150	999.99	0.100	5.130	0.100	4.650
GA (Binary)	992.44	999.99	999.99	999.99	999.99	999.99	999.99
GA (Binary)	721.96	999.99	999.99	999.99	999.99	999.99	999.99
GA (FP)	0.110	2.130	0.440	0.210	3.100	0.100	4.24
GA (FP)	998.99	999.94	999.91	999.07	999.65	999.96	999.98
GA (FP)	998.97	999.94	999.93	999.20	999.90	999.89	799.99
Literature	1.193	2.092	0.2236	0.100	3.003	0.100	4.056
Literature	1.193	2.092	0.2236	0.100	3.003	0.100	4.056
Literature	1.193	2.092	0.2236	0.100	3.003	0.100	4.056

Table 6.21: (Part II) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.22, 6.23 and 6.24)

Method	x <sub>11</sub>	x <sub>12</sub>	x <sub>13</sub>	x <sub>14</sub>	x <sub>15</sub>	x <sub>16</sub>	x <sub>17</sub>
Monte Carlo 1	48.17	1.32	911.54	978.25	125.93	92.57	628.68
Monte Carlo 1	419.91	820.36	641.79	768.99	995.88	119.53	735.38
Monte Carlo 1	225.59	932.22	396.87	946.95	939.75	941.38	470.30
Min-Max (OS)	47.418	47.418	47.418	47.418	47.418	47.418	47.418
Min-Max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
Min-Max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GA (Binary)	0.660	0.340	10.270	0.430	8.170	1.370	0.100
GA (Binary)	999.99	653.17	999.99	999.99	999.99	999.99	999.99
GA (Binary)	999.99	999.99	999.99	999.99	999.99	999.99	999.99
GA (FP)	1.04	0.710	6.45	0.39	7.34	1.24	2.11
GA (FP)	999.89	999.75	999.99	999.65	999.99	999.96	999.90
GA (FP)	999.98	999.83	999.96	999.92	999.93	999.99	999.99
Literature	0.2915	0.7880	7.036	0.100	7.043	0.3987	1.5547
Literature	0.2915	0.7880	7.036	0.100	7.043	0.3987	1.5547
Literature	0.2915	0.7880	7.036	0.100	7.043	0.3987	1.5547

Table 6.22: (Part III) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Tables 6.23 and 6.24)

Method	x <sub>18</sub>	x <sub>19</sub>	x <sub>20</sub>	x <sub>21</sub>	x <sub>22</sub>	x <sub>23</sub>	x <sub>24</sub>
Monte Carlo 1	899.65	167.02	466.18	264.20	953.07	390.65	326.46
Monte Carlo 1	610.54	585.89	830.03	448.94	527.13	470.69	975.88
Monte Carlo 1	634.71	447.23	526.03	550.32	916.07	959.82	935.28
Min-Max (OS)	47.418	47.418	47.418	47.418	47.418	47.418	47.418
Min-Max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
Min-Max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GA (Binary)	20.510	0.660	20.570	1.610	3.510	20.490	0.380
GA (Binary)	999.99	981.87	999.99	999.99	999.99	999.99	999.99
GA (Binary)	999.99	999.99	999.99	974.16	999.99	999.99	999.99
GA (FP)	19.33	0.100	11.05	2.00	1.21	15.58	1.79
GA (FP)	999.98	999.99	999.99	999.99	999.98	999.99	999.95
GA (FP)	999.97	999.87	999.99	999.96	999.98	999.97	999.89
Literature	7.2436	2.4735	8.1146	1.4121	3.3018	7.8096	3.4169
Literature	7.2436	2.4735	8.1146	1.4121	3.3018	7.8096	3.4169
Literature	7.2436	2.4735	8.1146	1.4121	3.3018	7.8096	3.4169

Table 6.23: (Part IV) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.24)

Method	x <sub>25</sub>	x <sub>26</sub>	x <sub>27</sub>	x <sub>28</sub>	x <sub>29</sub>
Monte Carlo 1	15.40	20.94	127.92	577.68	60.77
Monte Carlo 1	688.30	644.94	749.22	944.70	706.74
Monte Carlo 1	902.98	293.74	998.16	989.32	388.58
Min-Max (OS)	47.418	47.418	197.42	197.42	197.42
Min-Max (OS)	49.601	63.74	199.60	199.60	199.60
Min-Max (OS)	63.744	49.601	199.60	199.60	199.60
GA (Binary)	25.69	5.130	2.650	7.690	40.970
GA (Binary)	999.99	999.99	999.99	999.99	999.99
GA (Binary)	999.99	999.99	999.99	999.99	999.99
GA (FP)	20.10	4.01	4.92	11.24	21.67
GA (FP)	999.99	999.98	999.98	999.99	999.99
GA (FP)	999.99	999.98	999.98	999.91	999.99
Literature	8.7839	4.9573	14.2127	14.4496	14.7895
Literature	8.7839	4.9573	14.2127	14.4496	14.7895
Literature	8.7839	4.9573	14.2127	14.4496	14.7895

Table 6.24: (Part V) Comparison of results computing the ideal vector of example 6 from Chapter 5 (design of a 200-bar plane truss). OS stands for Osyczka's Multiobjective Optimization System.

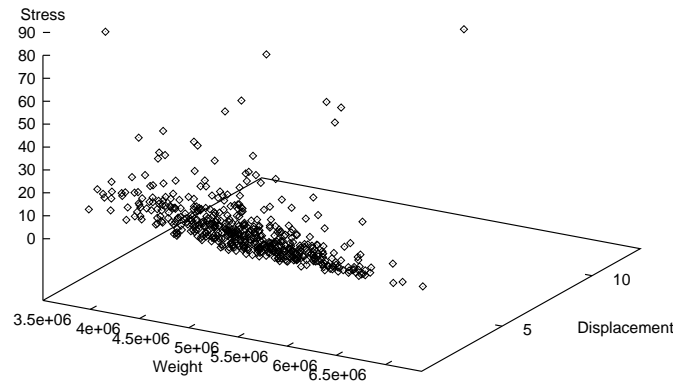


Figure 6.169: Initial feasible region for Monte Carlo method solving the sixth example.

Once again, the large size of the search space of this problem makes it impractical to show graphically the feasible region. The problem has 3 objective functions, 29 decision variables and 200 constraints. Since the range of each decision variable goes from 0.1 to 999.99, considering increments of 0.01 we would have to perform  $100000^{29}$  iterations to generate the feasible region (about  $1 \times 10^{145}$  iterations). Considering the amount of time needed to generate all the constraint information and to analyze the structure, this goal seems infeasible in a reasonable amount of time. Nevertheless, as in the previous example, I will provide 3-dimensional representations of the objective function space with each method, to get a good idea of how the feasible zone appears, and where the Pareto front is. As before, to solve this problem, it was necessary to add a module to each program in order to analyze the space truss generated by each algorithm. This module uses the matrix factorization method included in Gere and Weaver [202] together with the stiffness method [202] [203] to analyze the structure.

The ideal vector of this problem was computed using Monte Carlo Methods 1 and 2 (generating 500 points) presented in Chapter 4, and a GA (with a population of 500 chromosomes running during 100 generations) using binary and floating point representation, with the procedure described in Chapter 4 to adjust its parameters. The corresponding results are shown in Tables 6.20, 6.21 6.22, 6.23 and 6.24 including the best results reported in the literature [193]. Notice that the results presented by Belegundu violate 34 constraints of the problem, which means that his solution is not valid. This explains why the GA could not achieve such a low weight using floating point representation. In fact, in Belegundu's thesis [193] he even provides a better solution (with a total weight of 26261.05) but that violates 48 constraints. I chose to include a solution with a higher weight, but a lower number of violations. Nevertheless, the number of constraints violated is still high and the GA would not possibly converge towards such kind of solutions.

In this example, Monte Carlo method provided results that are within the average solutions provided by the GA-based techniques, which is remarkable, considering the large size of the search space. This reflects the problems of the GA to find reasonable trade-offs when the length of the chromosome string is too large (493 genes in this case). Also the high amount of constraints (200 total) makes this problem easier for mathematical programming techniques than for the GA using a penalty function. The performance of Osyczka's multiobjective optimization system is extremely good, but mainly because the initial guesses provided by the user were quite close to a Pareto solution. The main use of such techniques is precisely in cases in which we have a rough approximation of the solution, or a lot of knowledge about how the solution space appears, and we want to experiment within the boundaries of our result. Nevertheless, it should be pointed out that

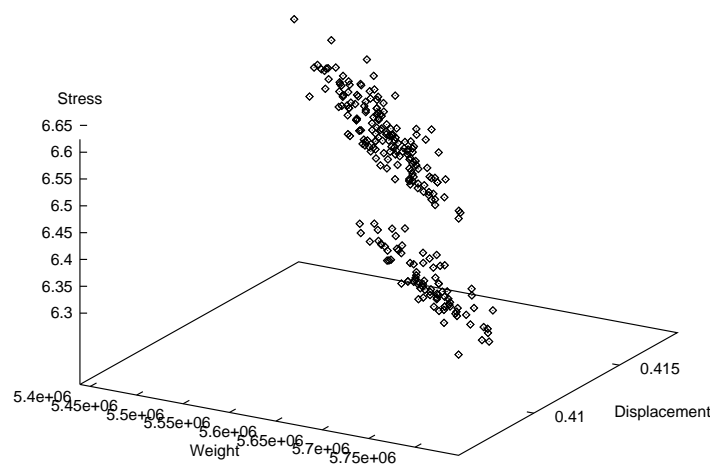


Figure 6.170: Example 6: The GA using a linear combination of the objectives with scaling, after 100 generations using binary representation.

my min-max method that uses weights was able to find a better overall result than any other technique (including mathematical programming methods) also for this example (using floating point representation) confirming its robustness as a numerical optimization tool. It should be noticed that in the following results, no graphs will be shown for more than 100 generations because of the extreme amount of CPU time required to iterate using the GA. To have an idea of such requirements, it is sufficient to be aware that each solution of the truss requires that the computer solves a system of  $N$  simultaneous linear equations, where  $N$  is the number of degrees of freedom of the structure (150 in this case). Even the GA in its simplest form will require to solve  $M \times G$  times the structure, where  $M$  is the population size and  $G$  the number of generations. That should give an idea of the heavy amount of calculations required for this example.

Since the Monte Carlo Methods previously mentioned and Osyczka's multiobjective optimization system do not generate the Pareto front, I will compare them with the GA-based approaches only in terms of the best overall result found.



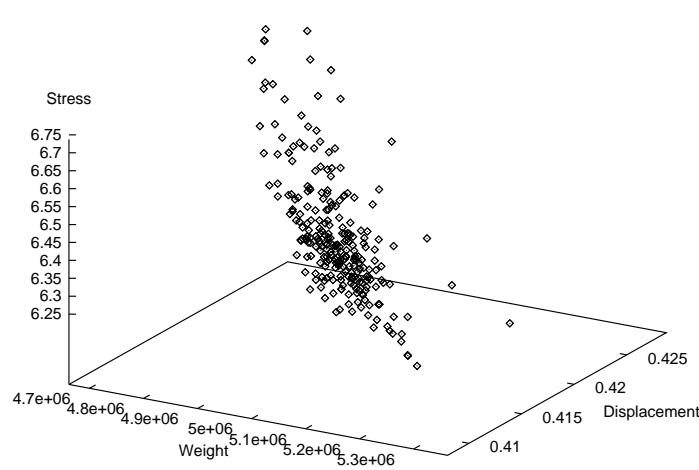


Figure 6.171: Example 6: The GA using a linear combination of the objectives with scaling, after 100 generations using floating point representation.

Besides those results, it is interesting to observe the initial distribution of points randomly generated by the method (see Figure 6.169). As can be seen from the tables previously shown (see Tables 6.20, 6.21, 6.22, 6.23 and 6.24) the objectives are highly conflicting, and therefore, it is very hard to come up with a good compromise. From the graph depicted in Figure 6.169 we can see that the optimum region would be located as close as possible to the ideal vector for each one of the axis. However, because of the great variation in the results, we will see that the common compromise will be to have higher values of weight to allow smaller stresses and displacements. This is true in the physical world, since larger cross-sectional areas allow less stress and displacement in the structure. It is also important observe through all the graphs corresponding to each example the ranges of the axis, because even when the clusters of points could all look alike, there will be a great variation in terms of the ranges.

We will start by analyzing the behavior of the **Lexicographic Method** at different stages of the search process. Figure 6.172 shows the distribution of

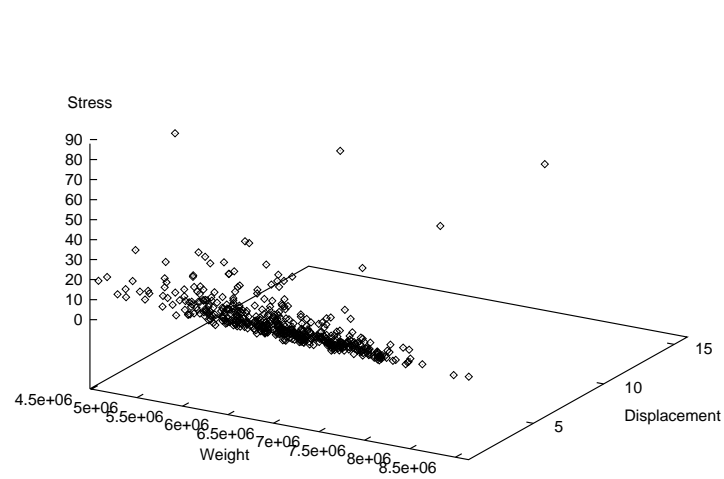


Figure 6.172: Example 6: Distribution of points using the Lexicographic Method with a binary representation at generation zero. Only points within the feasible region are displayed.

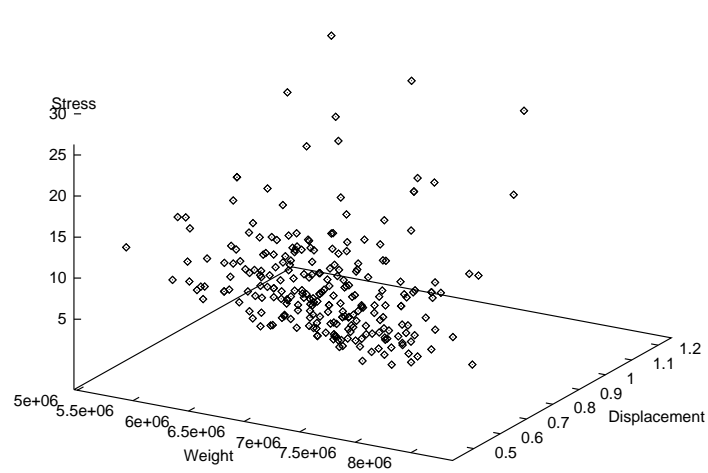


Figure 6.173: Example 6: Distribution of points using the Lexicographic Method with a binary representation at generation twenty. Only points within the feasible region are displayed.

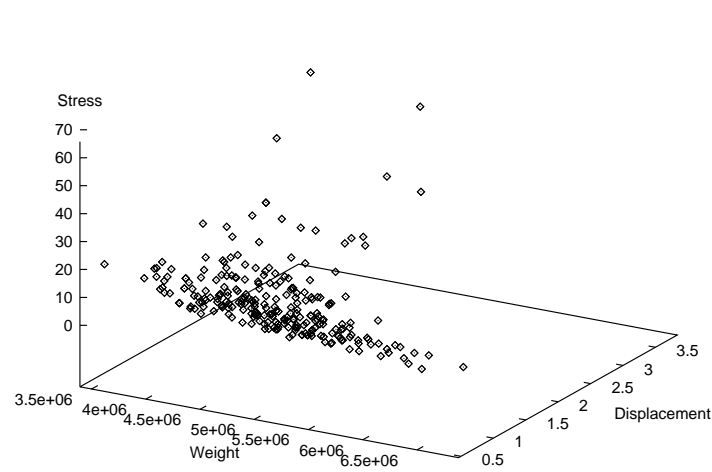


Figure 6.174: Example 6: Distribution of points using the Lexicographic Method with a floating point representation at generation twenty. Only points within the feasible region are displayed.

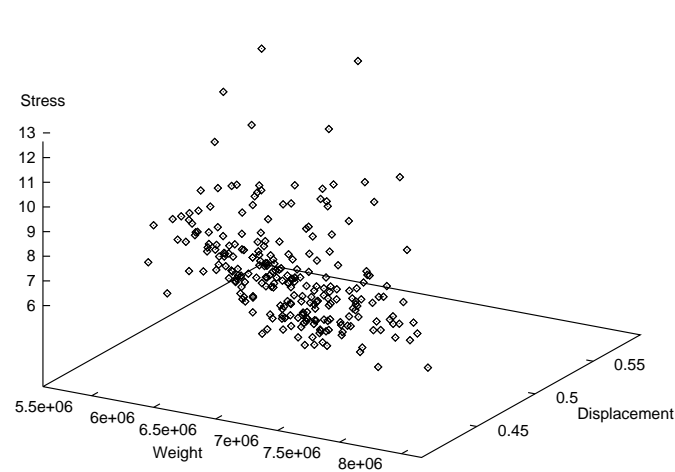


Figure 6.175: Example 6: Distribution of points using the Lexicographic Method with a binary representation at generation 100. Only points within the feasible region are displayed.

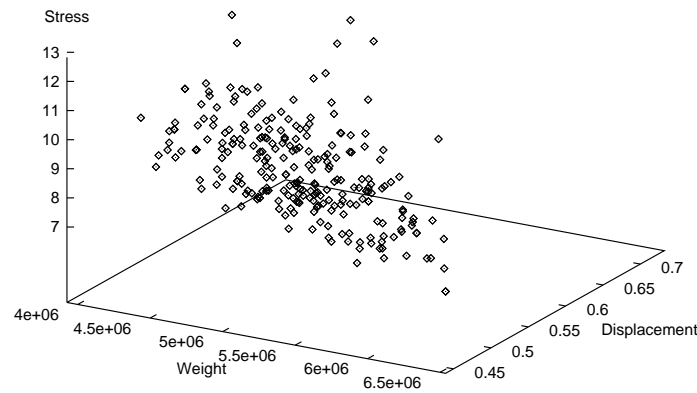


Figure 6.176: Example 6: Distribution of points using the Lexicographic Method with a floating point representation at generation 100. Only points within the feasible region are displayed.

points produced at generation zero using this method. This distribution is not very different than the initial distribution of Monte Carlo method. The main difference is in terms of the ranges: the graph for Monte Carlo method has a lower range for the weight at the initial population, but it has about the same ranges for the other two objective functions. The trade-off in both cases is more or less the same (weight and displacement have priority over stress), but there is a higher variation of solutions in the initial distribution of Monte Carlo method. Figure 6.173 shows the distribution of points at generation 20 using binary representation, and Figure 6.174 shows the corresponding distribution using floating point representation. The trade-offs provided by both graphs are different, because binary representation is favoring stress and displacement over weight, whereas floating point representation favors weight over stress and displacement. At this stage of the search, floating point representation offers a more uniform distribution of points than binary representation. After 100 generations, both graphs look remarkably similar, although floating point representation provides a better

trade-off solution and a slightly better distribution of points (see Figures 6.175 and 6.176). The final trade-off obtained using floating point representation was to slightly favor weight over stress and displacement. As before, I also compared the effect of a simple linear combination of objectives (addition or multiplication) using scaling. A linear combination of objectives produced a rather strange graph after 100 generations. When binary representation was used, the distribution was somewhat more uniform, without achieving total convergence of the population, although the ranges are very tight. Weight is slightly favored over displacement in this case (see Figure 6.170). When floating point representation was used, the final population formed two clusters of points with very tight ranges (see Figure 6.171). In this case, displacement was slightly favored over weight and stress. The best overall solution was found using floating point representation, because the population contains better trade-offs. However, the best overall results found with both representation schemes are poorer than those found with the Lexicographic method. Furthermore, when only a linear combination of objectives is used, convergence towards a single solution seems imminent (this can be easily seen from the tight ranges of the objective function values).

Schaffer's VEGA produces a semi-compact cluster of points after only 20 generations using binary representation (see Figure 6.177). Stress has been given priority over weight and displacement. Floating point representation produces a very similar distribution after 20 generations, but with a trade-off that favors weight over stress and displacement (see Figure 6.179). After 50 generations, the population looks more compact when a binary representation is used, and the ranges have been tightened, mainly with respect to displacement (see Figure 6.178). The compromise now is more clear: stress and displacement have priority over weight. The use of floating point representation produces a good

Method	$x_1$	$f_1$	$f_2$	$f_3$	$L_p(f)$
Ideal Vector		36167.73	0.370376	5.124250	0.000000
Monte Carlo 1	504.01	4475679.05	0.773293	12.173738	125.211428
Monte Carlo 2	370.14	5075790.44	0.561163	9.682691	140.745011
Min-max (OS)	49.601	641862.42	3.686123	88.04811	41.881840
GCM (OS)	49.601	641862.42	3.623485	85.86687	41.287050
WMM (OS)	49.601	641862.42	3.686123	88.04811	41.881840
PMM (OS)	47.418	617227.10	3.839831	91.97989	42.382994
NMM (OS)	49.601	641862.42	3.623485	85.86687	41.287050
GALC (B)	319.45	5388876.23	0.418137	6.623930	148.418423
GALC (FP)	41.85	4662186.53	0.426666	6.736934	128.371292
Lexicographic (B)	209.31	5106929.51	0.590409	11.446706	142.029185
Lexicographic (FP)	324.86	3925963.62	0.538302	9.329877	108.822923
VEGA (B)	238.54	5956689.85	0.546020	10.048603	165.131483
VEGA (FP)	786.24	4051105.81	0.662998	10.306576	112.810251
NSGA (B)	999.99	7020831.93	0.424951	6.678441	193.569332
NSGA (FP)	465.28	5369341.87	0.548805	8.334404	148.564917
MOGA (B)	201.94	3626863.83	0.489753	8.435848	100.247575
MOGA (FP)	91.26	2910316.85	0.722471	12.477403	81.852839
NPGA (B)	30.58	4028058.04	3.723483	73.607848	132.789464
NPGA (FP)	283.43	4453361.10	0.970288	5.468708	123.817748
Hajela (B)	95.47	1924166.83	1.362296	28.222939	59.387070
Hajela (FP)	137.90	4291090.29	0.874752	11.478412	120.245983
GAminmax1 (B)	13.26	1508966.74	2.213371	47.516121	53.970163
GAminmax1 (FP)	0.100	686362.29	2.223398	43.910769	30.549494
GAminmax2 (B)	910.90	9633841.60	0.370438	5.140362	265.368984
GAminmax2 (FP)	311.56	7816682.34	0.377604	5.508466	215.217609

Table 6.25: (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.26, 6.27, 6.28 and 6.29)

Method	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$
Monte Carlo 1	248.68	591.30	80.54	554.36	59.41	320.37	660.33
Monte Carlo 2	434.52	478.17	766.34	926.32	262.27	880.61	722.87
Min-max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GCM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
WMM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
PMM (OS)	47.418	47.418	47.418	47.418	47.418	47.418	47.418
NMM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GALC (B)	395.26	55.35	121.82	806.05	999.99	196.28	999.99
GALC (FP)	934.02	102.56	164.06	856.15	59.06	115.84	895.39
Lexicographic (B)	562.18	228.89	999.99	999.99	999.99	99.08	999.99
Lexicographic (FP)	490.93	261.98	716.80	378.18	493.62	172.48	947.73
VEGA (B)	536.52	999.99	481.29	469.10	999.99	999.99	456.86
VEGA (FP)	233.10	973.32	266.45	636.04	372.60	539.54	966.85
NSGA (B)	729.22	999.99	999.99	838.28	708.10	999.99	999.99
NSGA (FP)	394.50	916.59	429.33	900.76	215.59	970.86	895.58
MOGA (B)	23.71	316.63	999.99	872.65	778.18	976.24	999.99
MOGA (FP)	294.00	44.25	143.95	588.13	150.60	42.22	900.79
NPGA (B)	482.26	987.45	17.07	963.91	375.91	298.25	999.99
NPGA (FP)	376.94	961.69	912.80	485.72	41.95	10.60	438.87
Hajela (B)	5.78	5.99	20.06	41.37	999.99	6.74	228.58
Hajela (FP)	526.57	117.66	90.98	487.06	116.43	689.35	444.33
GAminmax1 (B)	4.93	13.52	21.46	81.38	999.99	10.80	91.17
GAminmax1 (FP)	102.77	15.17	8.95	60.59	13.89	0.54	72.27
GAminmax2 (B)	999.99	999.99	670.36	999.99	999.99	914.06	999.99
GAminmax2 (FP)	507.56	411.48	256.32	970.55	680.86	527.94	989.51

Table 6.26: (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.27, 6.28 and 6.29)

Method	x <sub>9</sub>	x <sub>10</sub>	x <sub>11</sub>	x <sub>12</sub>	x <sub>13</sub>	x <sub>14</sub>	x <sub>15</sub>
Monte Carlo 1	201.44	938.64	216.95	480.92	233.70	433.72	349.93
Monte Carlo 2	787.24	677.48	91.48	162.18	949.91	931.35	562.31
Min-max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GCM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
WMM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
PMM (OS)	47.418	47.418	47.418	47.418	47.418	47.418	47.418
NMM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GALC (B)	44.00	999.99	201.55	220.45	999.99	341.00	999.99
GALC (FP)	211.55	959.81	179.72	122.01	998.51	290.49	991.09
Lexicographic (B)	500.55	417.79	396.29	688.26	941.10	686.64	923.30
Lexicographic (FP)	379.78	493.43	483.86	366.32	377.72	176.23	299.43
VEGA (B)	999.99	529.65	999.99	835.83	804.09	645.22	999.99
VEGA (FP)	958.67	738.16	169.25	285.13	598.61	26.18	425.17
NSGA (B)	305.61	724.70	669.27	360.02	999.99	771.59	999.99
NSGA (FP)	630.80	689.38	584.35	821.89	875.03	282.05	908.43
MOGA (B)	993.94	999.99	407.08	77.73	985.89	348.38	999.99
MOGA (FP)	294.47	788.77	292.17	206.14	549.07	783.29	973.79
NPGA (B)	401.86	989.95	15.94	467.94	78.66	854.04	89.81
NPGA (FP)	629.11	885.84	149.11	874.80	982.86	604.00	634.80
Hajela (B)	10.20	177.92	37.31	23.63	238.07	7.09	194.55
Hajela (FP)	138.28	560.16	443.88	564.05	284.91	84.72	295.44
GAminmax1 (B)	15.99	54.46	19.11	19.69	133.50	12.17	185.38
GAminmax1 (FP)	4.32	101.02	19.86	24.66	91.77	91.34	159.49
GAminmax2 (B)	999.99	999.99	999.99	971.81	999.99	999.99	999.99
GAminmax2 (FP)	405.85	999.86	896.94	910.15	992.17	661.75	965.52

Table 6.27: (Part III) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Tables 6.28 and 6.29)



Method	x <sub>16</sub>	x <sub>17</sub>	x <sub>18</sub>	x <sub>19</sub>	x <sub>20</sub>	x <sub>21</sub>	x <sub>22</sub>
Monte Carlo 1	605.46	650.30	89.46	396.33	357.04	526.03	883.42
Monte Carlo 2	242.57	48.88	315.34	875.24	444.38	492.37	935.04
Min-max (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GCM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
WMM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
PMM (OS)	47.418	47.418	47.418	47.418	47.418	47.418	47.418
NMM (OS)	49.601	49.601	49.601	49.601	49.601	49.601	49.601
GALC (B)	192.22	365.82	999.99	410.63	999.99	225.98	772.74
GALC (FP)	221.00	398.75	993.63	407.73	971.32	304.32	258.93
Lexicographic (B)	343.48	822.80	902.24	165.95	778.62	242.05	552.68
Lexicographic (FP)	89.84	113.99	880.21	686.89	801.43	730.66	283.87
VEGA (B)	728.28	993.66	999.99	26.35	836.59	999.99	999.99
VEGA (FP)	921.41	446.62	738.61	532.28	866.27	477.85	854.78
NSGA (B)	999.99	969.01	999.99	730.88	895.97	999.99	652.47
NSGA (FP)	711.12	358.32	678.77	310.20	738.32	396.34	532.93
MOGA (B)	262.00	205.07	999.99	707.30	990.61	218.69	465.08
MOGA (FP)	212.44	103.93	890.15	602.09	987.54	201.35	32.30
NPGA (B)	126.10	648.53	834.68	999.99	999.99	381.70	970.05
NPGA (FP)	456.01	771.41	288.38	337.69	953.04	387.24	250.90
Hajela (B)	23.07	15.20	416.22	56.91	422.00	39.39	41.97
Hajela (FP)	725.96	463.73	959.95	71.77	848.02	184.86	661.01
GAminmax1 (B)	37.05	4.56	205.69	78.08	159.84	25.63	41.85
GAminmax1 (FP)	25.07	1.65	223.10	30.81	383.57	46.74	90.27
GAminmax2 (B)	999.99	999.99	999.99	999.99	999.99	999.99	999.99
GAminmax2 (FP)	883.82	983.68	968.51	813.06	959.20	961.54	849.51

Table 6.28: (Part IV) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective). (Continued in Table 6.29)

Method	x <sub>23</sub>	x <sub>24</sub>	x <sub>25</sub>	x <sub>26</sub>	x <sub>27</sub>	x <sub>28</sub>	x <sub>29</sub>
Monte Carlo 1	464.17	186.46	591.06	599.54	907.51	818.03	211.24
Monte Carlo 2	384.91	305.80	988.40	528.76	742.48	939.33	668.10
Min-max (OS)	49.601	49.601	49.601	49.601	199.60	199.60	213.74
GCM (OS)	49.601	49.601	63.744	49.601	199.60	199.60	199.60
WMM (OS)	49.601	49.601	49.601	49.601	199.60	199.60	213.74
PMM (OS)	47.418	47.418	47.418	47.418	197.42	197.42	197.42
NMM (OS)	49.601	49.601	63.744	49.601	199.60	199.60	199.60
GALC (B)	999.99	550.28	999.99	610.24	999.99	999.99	999.99
GALC (FP)	956.89	845.72	990.23	893.61	980.45	958.03	998.39
Lexicographic (B)	999.99	999.99	917.21	617.42	343.89	999.99	857.43
Lexicographic (FP)	954.03	938.10	738.21	325.21	950.14	791.35	847.54
VEGA (B)	715.82	555.71	999.99	938.54	999.99	999.99	999.99
VEGA (FP)	981.52	749.77	871.18	602.99	798.65	620.77	844.67
NSGA (B)	999.99	723.36	999.99	999.99	738.68	999.99	654.16
NSGA (FP)	634.31	304.25	931.62	924.14	546.39	898.56	699.44
MOGA (B)	999.99	669.93	999.99	999.99	668.86	999.99	999.99
MOGA (FP)	920.64	87.60	967.83	207.61	812.97	968.07	998.36
NPGA (B)	15.78	286.15	797.16	3.79	407.56	839.46	413.57
NPGA (FP)	809.75	158.71	646.77	513.51	61.09	617.50	205.76
Hajela (B)	533.19	124.87	394.24	112.04	160.36	472.10	665.32
Hajela (FP)	671.18	781.38	199.67	653.79	416.76	632.24	215.40
GAminmax1 (B)	260.63	76.28	242.75	74.56	97.55	203.68	344.79
GAminmax1 (FP)	208.84	108.13	215.59	52.45	101.15	240.78	223.00
GAminmax2 (B)	999.99	999.99	999.99	999.99	999.99	999.99	999.99
GAminmax2 (FP)	991.08	980.92	975.94	965.24	994.13	988.03	998.31

Table 6.29: (Part V) Comparison of the best overall solution found by each one of the methods included in MOSES for the sixth example (design of a 200-bar plane truss). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.33 (equal weight for every objective).

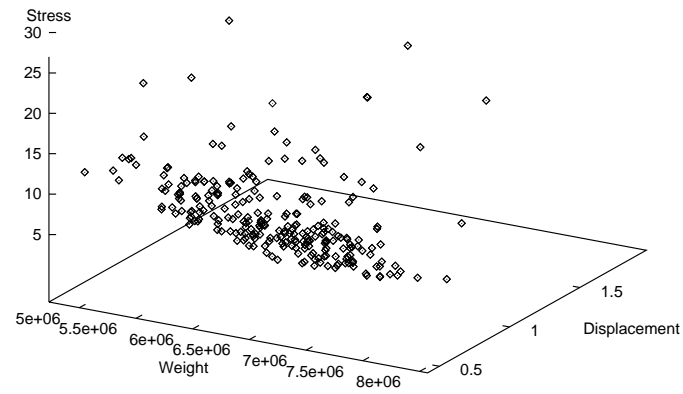


Figure 6.177: Example 6: Distribution of points using VEGA with a binary representation at generation twenty.

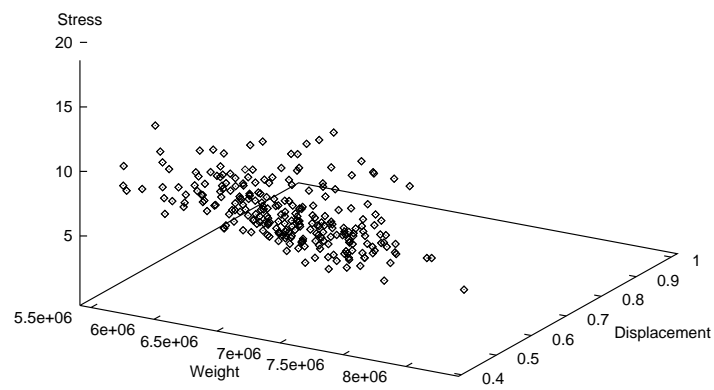


Figure 6.178: Example 6: Distribution of points using VEGA with a binary representation at generation fifty.

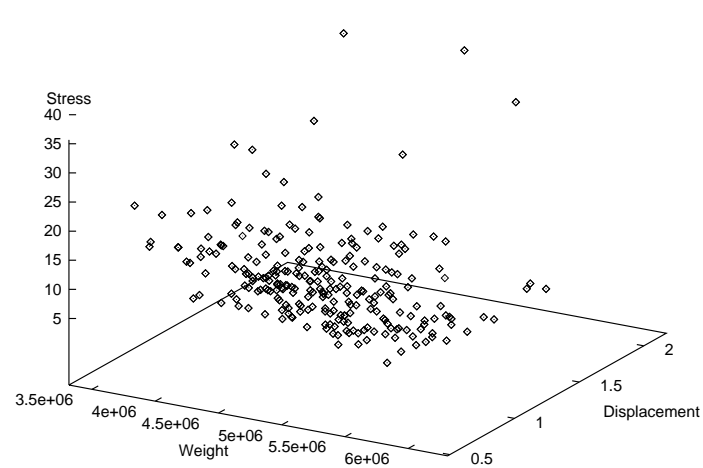


Figure 6.179: Example 6: Distribution of points using VEGA with floating point representation at generation twenty.

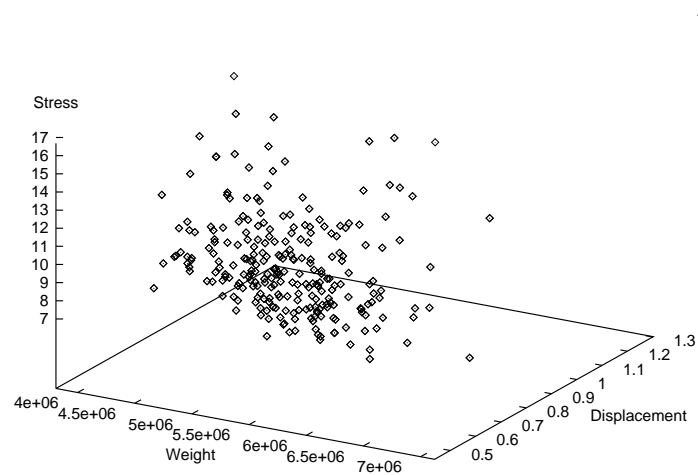


Figure 6.180: Example 6: Distribution of points using VEGA with floating point representation at generation fifty.

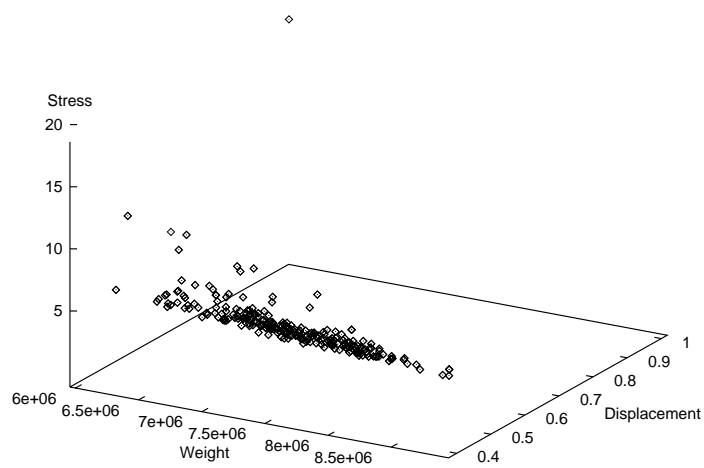


Figure 6.181: Example 6: Distribution of points using VEGA with binary representation at generation 100.

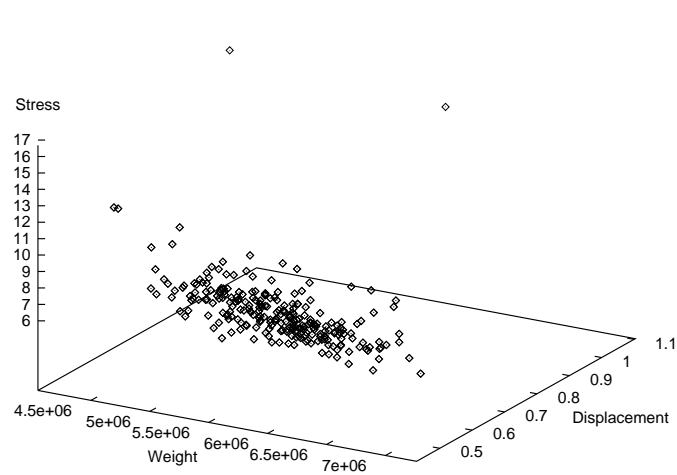


Figure 6.182: Example 6: Distribution of points using VEGA with floating point representation at generation 100.

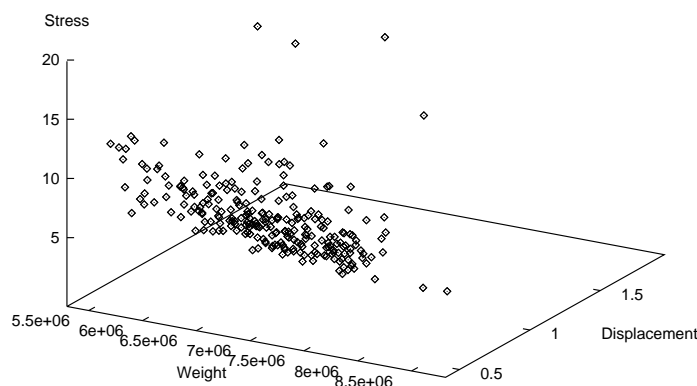


Figure 6.183: Example 6: Distribution of points using NSGA with binary representation at generation twenty.

sketch of the Pareto front, with tight ranges for the weight and displacement but with a broader range for the stress (see Figure 6.180). The ranges of the axis give the impression of a more sparse distribution in this latter case, but the population is quite compact. After 100 generations, a binary representation provides an almost linear distribution of points, forming what seems to be the Pareto front (see Figure 6.181). Floating point representation produces a front somewhat more sparse, but wider along the  $x$  and  $z$  axis, which is the reason why it produces a better overall solution (see Figure 6.182). Comparing these 2 previous graphs, we can easily see that this is another example of how floating point representation is able to keep a better diversity of the population in certain problems, mainly when the size of the search space (and, consequently, the length of the chromosome strings) is fairly large.

After 20 generations, my version of Srinivas' NSGA that uses binary representation produces a rather compact cluster of points representing solutions in which weight is favored over stress and displacement (see Figure 6.183). After

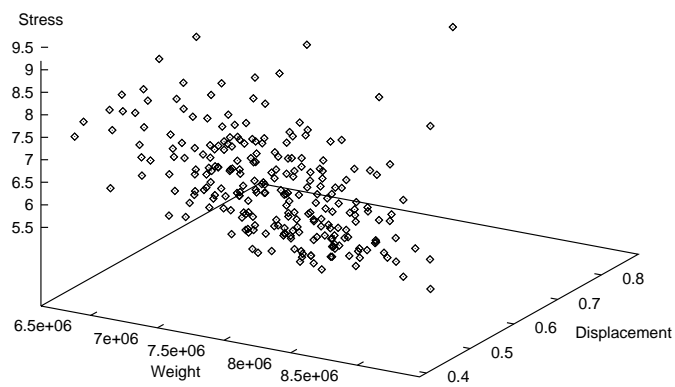


Figure 6.184: Example 6: Distribution of points using NSGA with binary representation at generation 50.

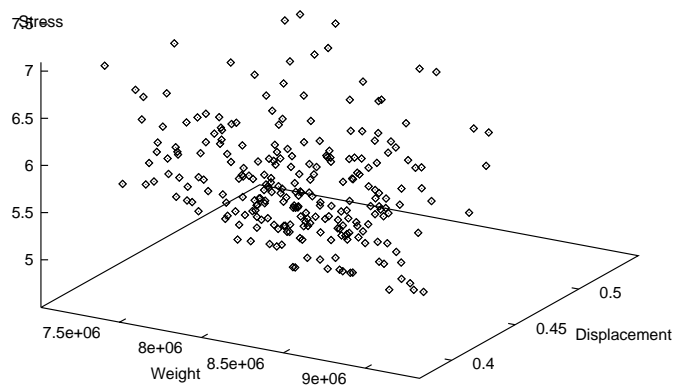


Figure 6.185: Example 6: Distribution of points using NSGA with binary representation at generation 100.

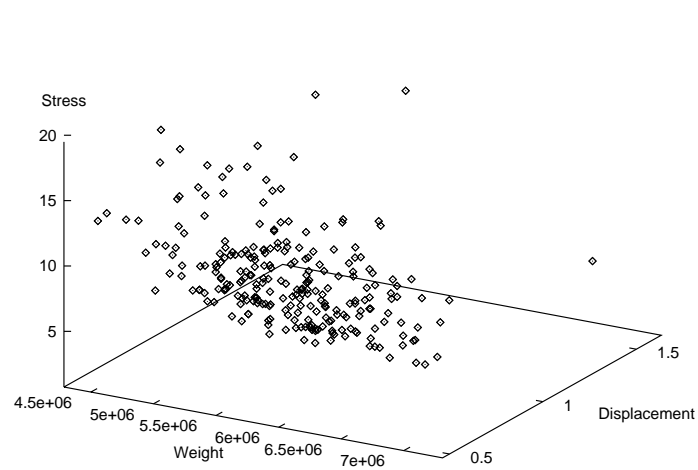


Figure 6.186: Example 6: Distribution of points using NSGA with floating point representation at generation 20.

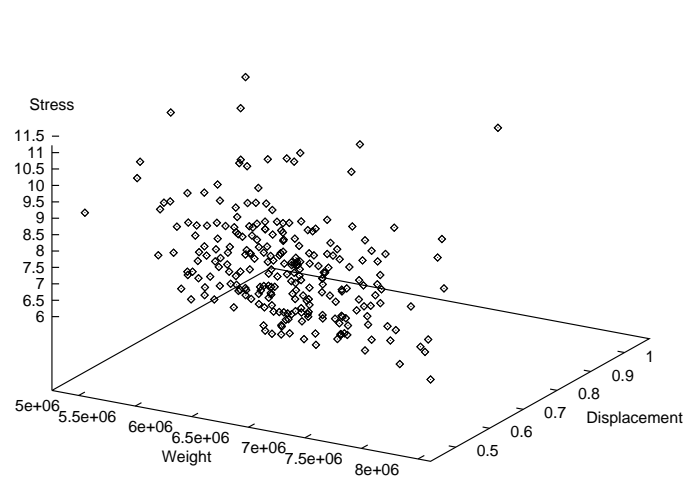


Figure 6.187: Example 6: Distribution of points using NSGA with floating point representation at generation 50.



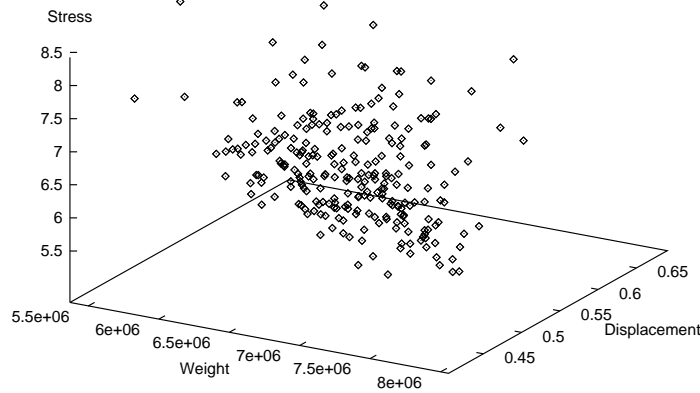


Figure 6.188: Example 6: Distribution of points using NSGA with floating point representation at generation 100.

50 generations, the population seems more sparse, but the ranges are narrower, indicating that the cluster is very compact (see Figure 6.184). At this point, stress and displacement have been favored over weight. After 100 generations the population looks more sparse, although the ranges of the axis are even narrower than before (see Figure 6.185). The compromise adopted by this method seems clear now: stress and displacement have priority over weight. The use of floating point representation produces better results in this example. After 20 generations, the distribution of points looks sparse, and weight has been favored over stress and displacement (see Figure 6.186). The Pareto front seems more clear after 50 generations, in which better trade-offs are made, allowing slightly higher weights to decrease stress and displacement (see Figure 6.187). This process continues after 100 generations, where weights have been newly increased to decrease stress and displacement (see Figure 6.188). At this point, the population looks more sparse, but that is because the ranges of the axis are narrower. Floating point representation provided a much better overall solution, because the population after

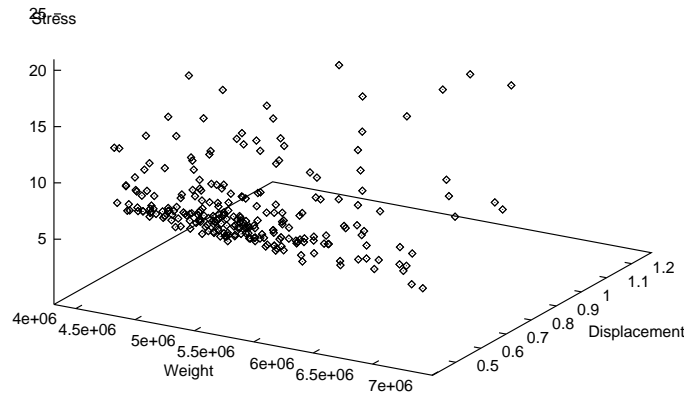


Figure 6.189: Example 6: Distribution of points using MOGA with binary representation at generation 20.

100 generations keeps better compromises between weight and displacement. In this case, I did not run tests up to 500 generations because of the excessive CPU required. Finally, as in previous examples, there was no significant change in the results when the value of  $\sigma_{share}$  was modified.

Fonseca's MOGA gives very good compromises even after only 20 generations using binary representation, favoring weight over stress and displacement, but with a better compromise than the previous methods (see Figure 6.189). At this stage, there is a solid group of points in the compromise zone, but there are still several dominated points. After 100 generations, however, the Pareto front is very clear, with slightly increased stresses and lower displacements (see Figure 6.190). Weights remain about the same as before, but with a narrower range of variation. Because of the low weights achieved while keeping reasonable low stresses and displacements, the best overall solution found using binary representation is quite good as compared to what the previous GA-based methods analyzed so far could do in this problem. Floating point representation provides,

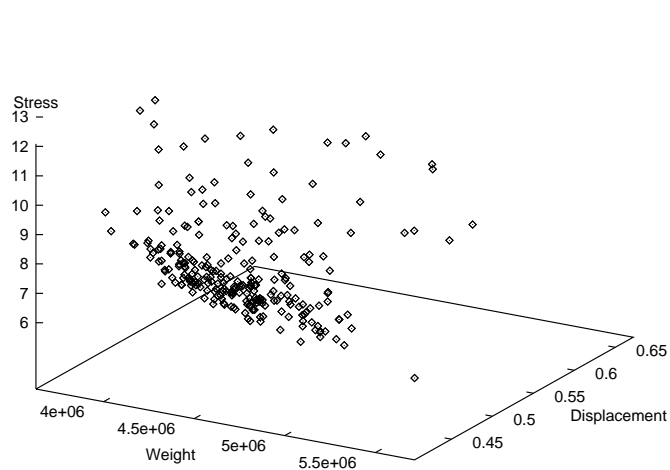


Figure 6.190: Example 6: Distribution of points using MOGA with binary representation at generation 100.

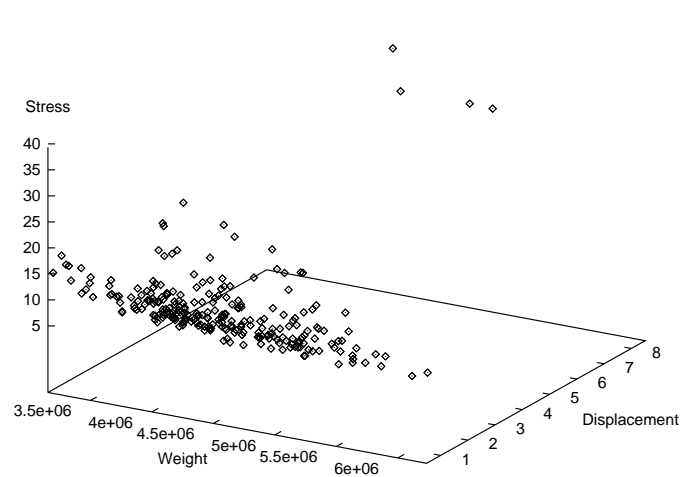


Figure 6.191: Example 6: Distribution of points using MOGA with floating point representation at generation 20.

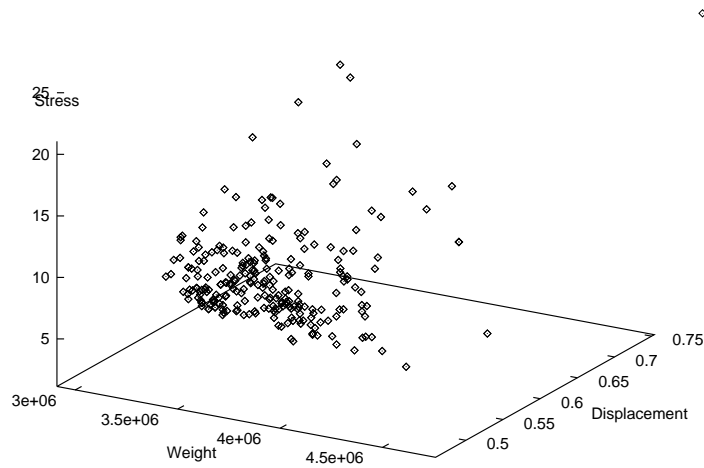


Figure 6.192: Example 6: Distribution of points using MOGA with floating point representation at generation 100.

even better compromises than binary representation. After only 20 generations weight has taken priority over stress and displacement, and a very compact cluster of points has been gathered (see Figure 6.191). After 100 generations, however, the Pareto front looks more like that found using binary representation, although in this case the compromise is better, because weights are kept lower than before without significantly affecting stress and displacement (see Figure 6.192). Because of the lower weights achieved using floating point representation, the best overall result found using this representation scheme is better than when binary representation was used. I also used a value of  $\sigma_{share} = 0.1$  as in NSGA to generate these graphs, but changing this value did not affect the results produced by the algorithm. There was no attempt to run this algorithm for more than 100 generations because of the high demands of CPU time required by this method.

As in previous cases, NPGA shows a remarkably consistent behavior, that corroborates its robustness as a multiobjective optimization technique. After 20 generations using binary representation, the Pareto front is already visible, and

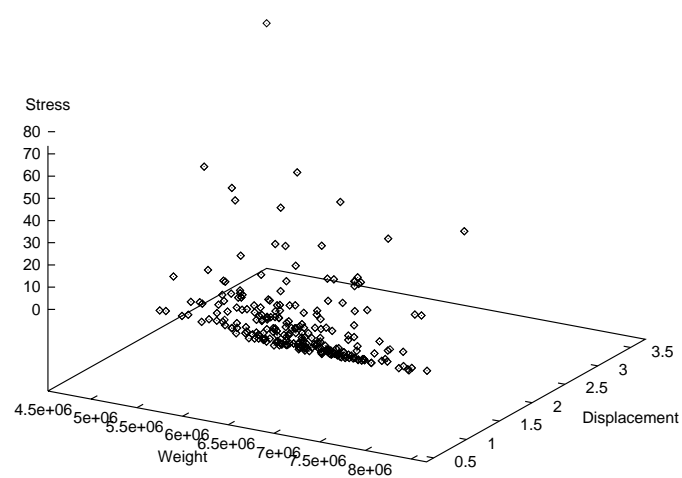


Figure 6.193: Example 6: Distribution of points using NPGA with binary representation at generation 20.

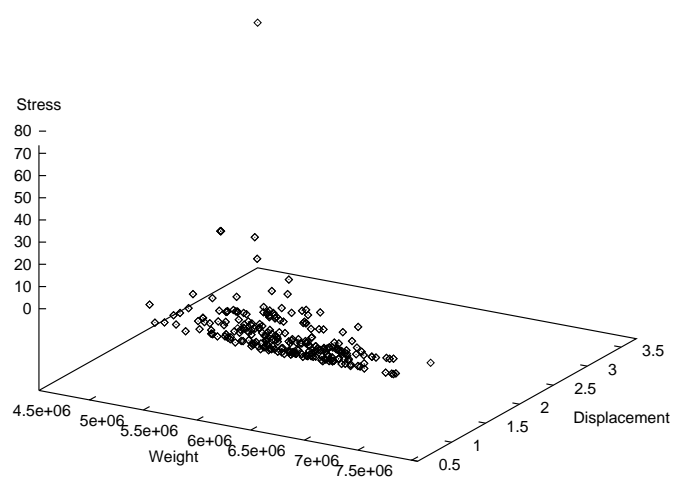


Figure 6.194: Example 6: Distribution of points using NPGA with binary representation at generation 50.

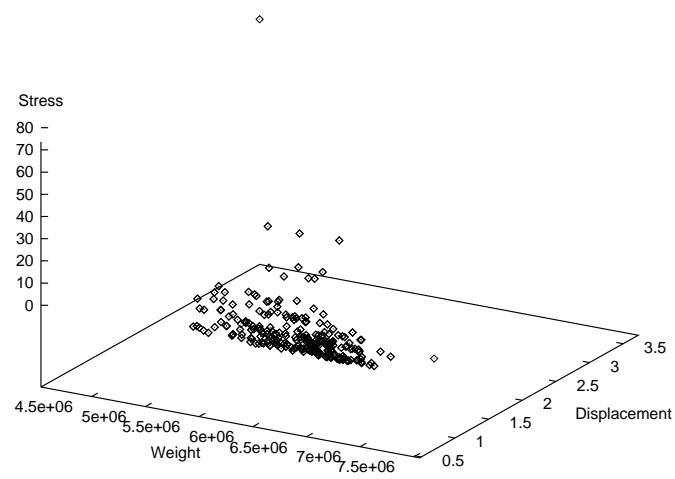


Figure 6.195: Example 6: Distribution of points using NPGA with binary representation at generation 100.

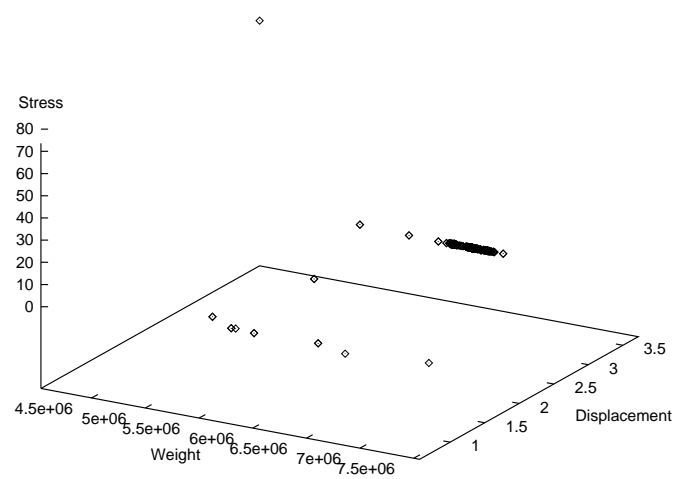


Figure 6.196: Example 6: Distribution of points using NPGA with binary representation at generation 100 with  $\sigma_{share} = 1.0$ .

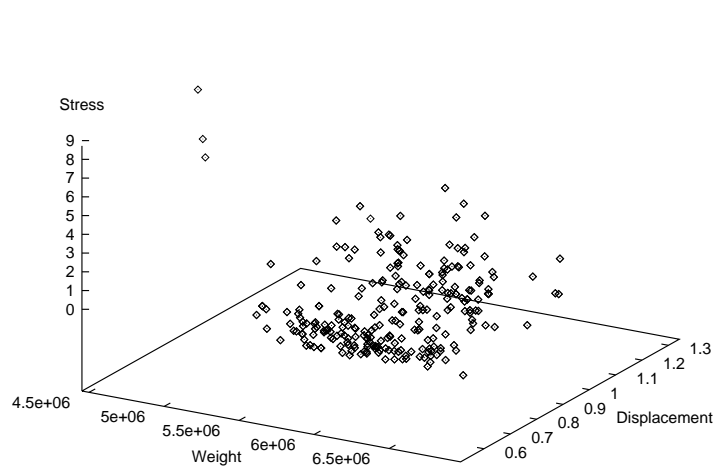


Figure 6.197: Example 6: Distribution of points using NPGA with floating point representation at generation 100.

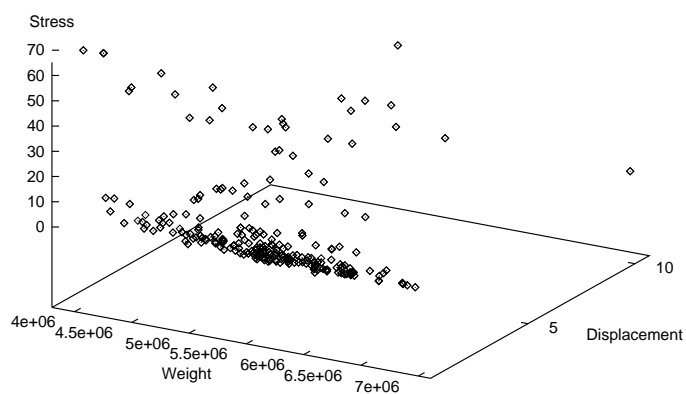


Figure 6.198: Example 6: Distribution of points using NPGA with floating point representation at generation 20.

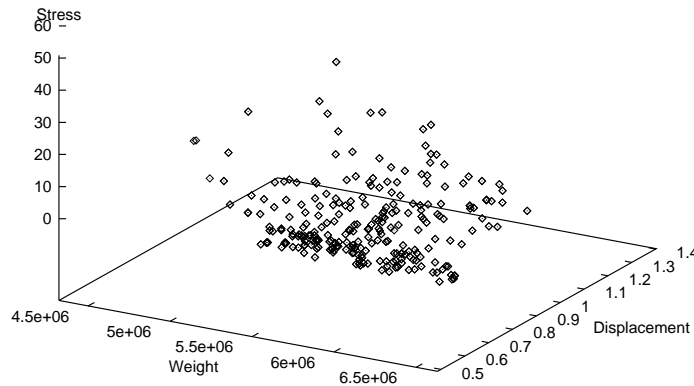


Figure 6.199: Example 6: Distribution of points using NPGA with floating point representation at generation 50.

the compromise is to favor weight over stress and displacement (see Figure 6.193). After 50 generations, there are fewer dominated points in the population, and the Pareto front is quite clear (see Figure 6.194). The compromise among the objectives remains the same. After 100 generations, the Pareto front looks like a plane formed by a very solid layer of points with exactly the same compromise as before (see Figure 6.195). Due to the high values of displacement and stress adopted, the best overall result is not as good as we would like (see Tables 6.20, 6.21, 6.22 and 6.23). When floating point representation is used, the results produced seem less consistent. After 20 generations, a cluster of points representing compromise solutions in which weight is highly favored over stress and displacement is generated (see Figure 6.198). However, after 50 generations, the trade-off has changed and displacement has been slightly favored over weight (see Figure 6.199). The distribution of points is, nevertheless, too sparse, when we compare it to that displayed at earlier stages of the search process. After 100 generations, the population is still too sparse, although there is certain resemblance of the Pareto front



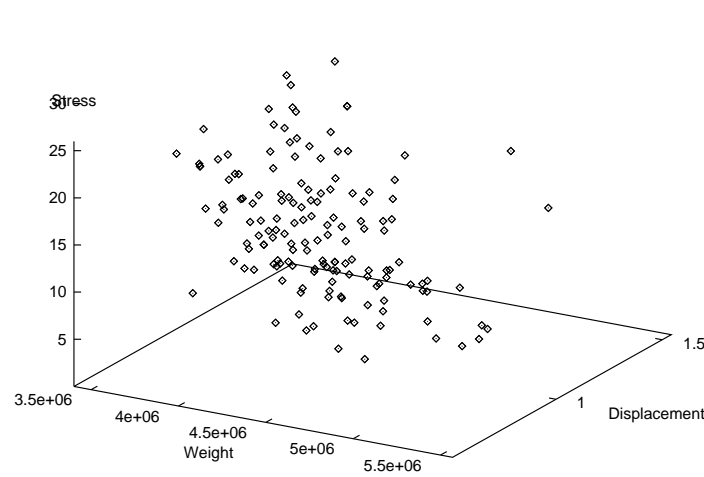


Figure 6.200: Example 6: Distribution of points using Hajela's method with binary representation at generation 20.

that binary representation produced (see Figure 6.197). It is, however, due to this sparse distribution of points that floating point representation produced a better overall solution than binary representation. As in previous examples, the value of  $\sigma_{share}$  plays a critical role in the performance of the algorithm, because if we increase it from 0.1 (same value used with the previous techniques that use niches) to 1.0, the entire population converges to a very small set of points after only 100 generations, producing very bad compromises among the objectives (see Figure 6.196).

Hajela's method provides a highly sparse distribution of points after 20 generations using binary representation, with trade-offs that strongly favor weight over displacement and stress (see Figure 6.200). After 50 generations, this trend is even more noticeable, and lower weights are being generated while stress increases and displacement remains the same as at earlier generations (see Figure 6.201). After 100 generations the population has formed a very compact plane that constitutes the Pareto front, with compromises that highly favor weight over stress

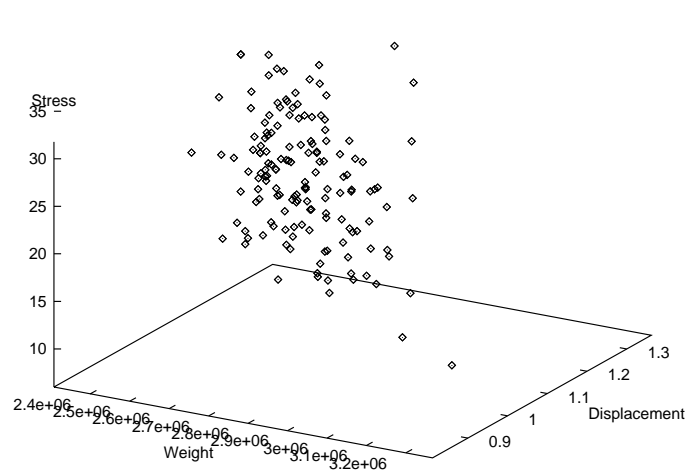


Figure 6.201: Example 6: Distribution of points using Hajela's method with binary representation at generation 50.

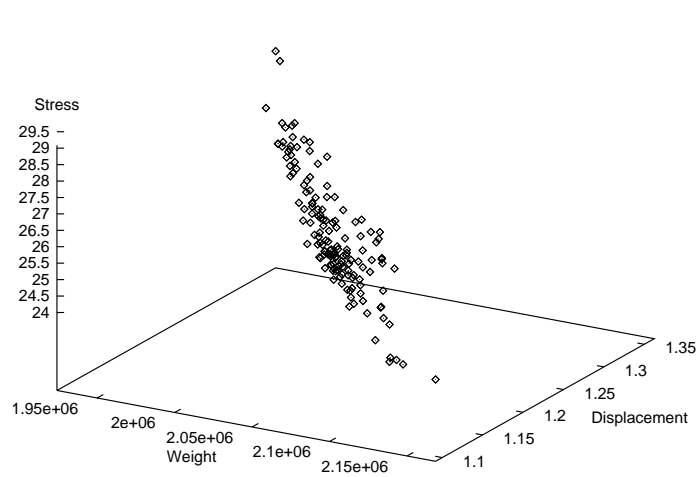


Figure 6.202: Example 6: Distribution of points using Hajela's method with binary representation at generation 100.

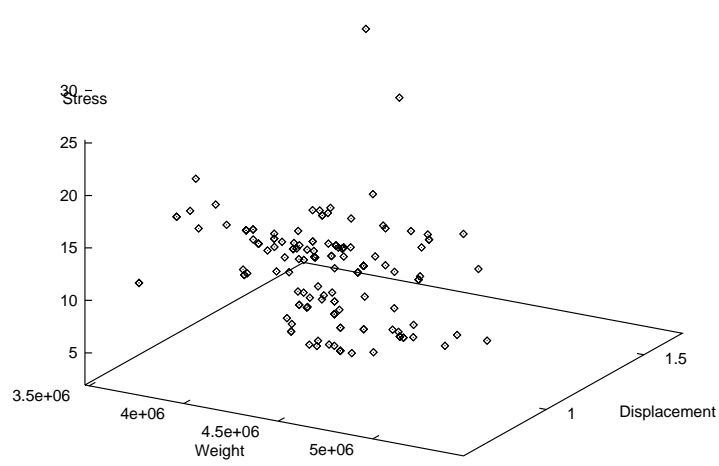


Figure 6.203: Example 6: Distribution of points using Hajela's method with floating point representation at generation 20.

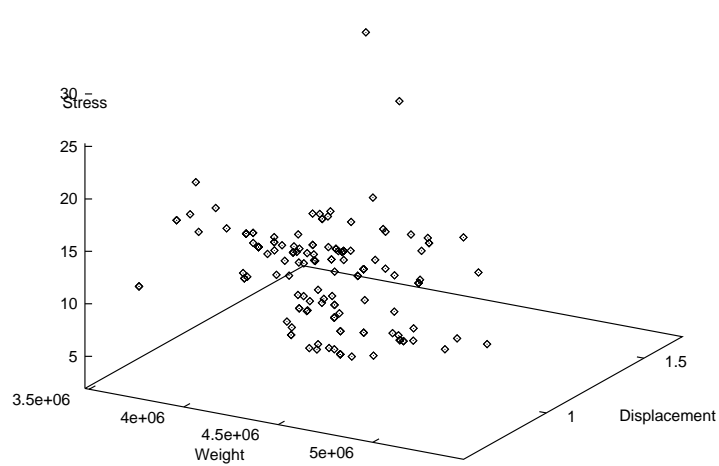


Figure 6.204: Example 6: Distribution of points using Hajela's method with floating point representation at generation 20.

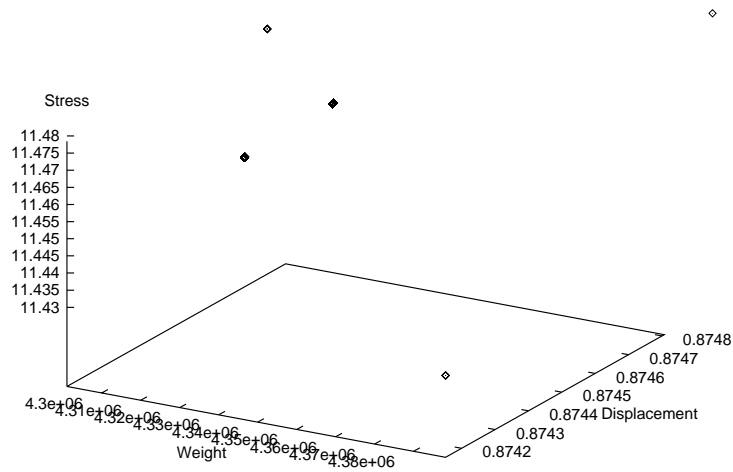


Figure 6.205: Example 6: Distribution of points using Hajela's method with floating point representation at generation 100.

and displacement (see Figure 6.202). Notice that since the weights produced are extremely low (compared to the values produced by previous methods), the best overall result that this method generates using binary representation is excellent. Floating point representation has a disappointing behavior in this example. After only 20 generations we can see a notorious lack of points in the graph, indicating redundancy in the solutions encoded by the population (see Figure 6.203). The compromise is similar as when binary representation is used, given weight preference over stress and displacement. After 50 generations only a few points remain in the graph, which means that the GA has practically converged to a single solution (see Figure 6.204). In this case, the weight and stress have increased, allowing a lower displacement. After 100 generations, total convergence is a fact as can be seen in the corresponding graph (see Figure 6.205). The best overall solution found using floating point representation is much poorer than the one found using binary representation, because convergence in the former case took place towards a zone of not very good compromises (displacement has priority

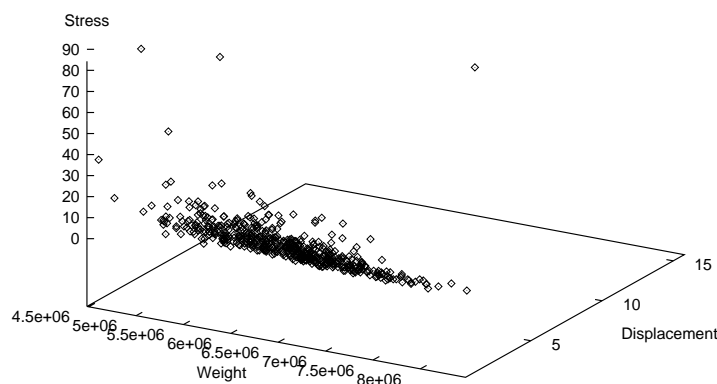


Figure 6.206: Example 6: Distribution of points using my method based on the min-max algorithm with binary representation at generation zero.

over stress and weight). Again, the set of weights and the niching parameters used by this technique play a crucial role on its performance.

Before showing the results produced with my first method based on the min-max approach, I want to show the distribution of points at generation zero, since in this case the algorithm ensures that only feasible solutions are generated. Figure 6.206 shows such initial distribution. The cluster of points at generation zero is gathered around a straight line that delineates a trade-off in which weight is favored over stress and displacement. After 100 generations, and using 20 different weights with binary representation, the Pareto front is perfectly visible (see Figure 6.207). The compromise still favors weight over stress and displacement, but with a better trade-off than that produced by all the previous GA-based methods. Floating point representation produces a similar contour, but with a much better trade-off among the 3 objectives (see Figure 6.208). The best overall result produced in this latter case has the lowest deviation from all techniques (including Osyczka's system with very good guesses), confirming the robustness of this

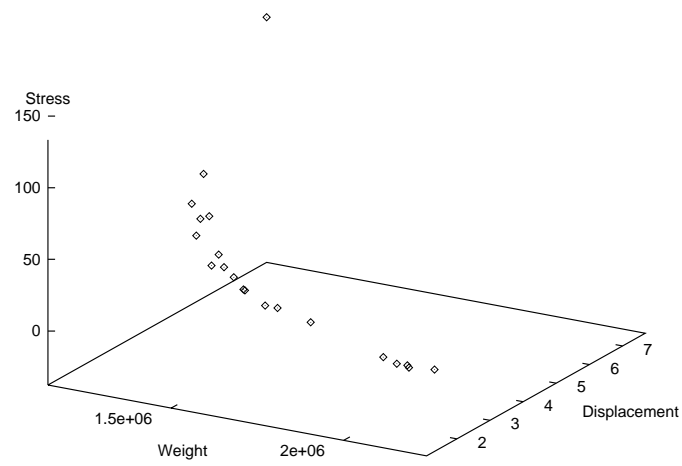


Figure 6.207: Example 6: Distribution of points using my method based on the min-max algorithm with binary representation at generation 100.

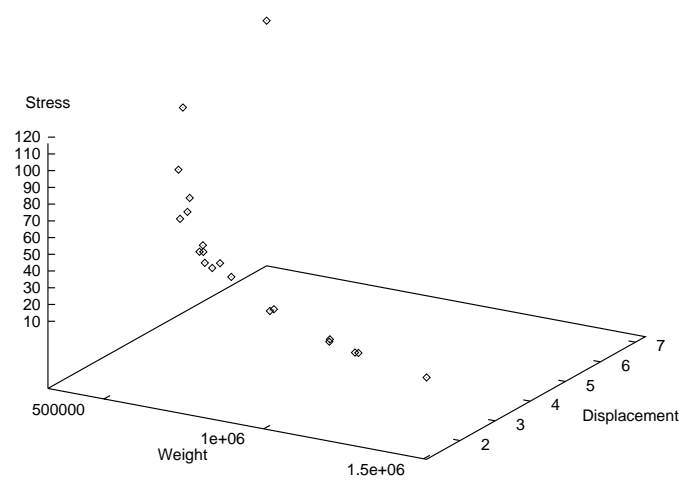


Figure 6.208: Example 6: Distribution of points using my method based on the min-max algorithm with floating point representation at generation 100.

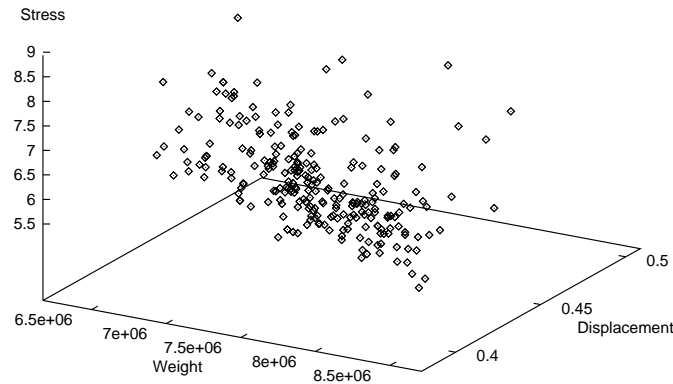


Figure 6.209: Example 6: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 20.

technique as a numerical optimization tool. It should be reminded, however, that the performance of this technique is highly dependent upon the weights provided by the user. Nevertheless, the results provided by this technique make it worth experimenting with the weights mainly if we consider that it turns out to be less difficult to tune up this parameter in practice than what one could expect. This has been shown with the examples solved in this thesis in which the search space has a very difficult shape for most GA-based techniques, and the set of weights used to produce the results reported was chosen only once.

This example produces the worst observed behavior for my second technique based on the min-max algorithm. After 20 generations using binary representation, the population has a good distribution, and the Pareto front is clearly visible (see Figure 6.209). The compromise adopted has been to favor displacement and stress over weight. After 100 generations, however, the population is starting to converge to a unique point, and the algorithm is having a lot of trouble trying to avoid that (see Figure 6.210). The ranges at this point look very tight,

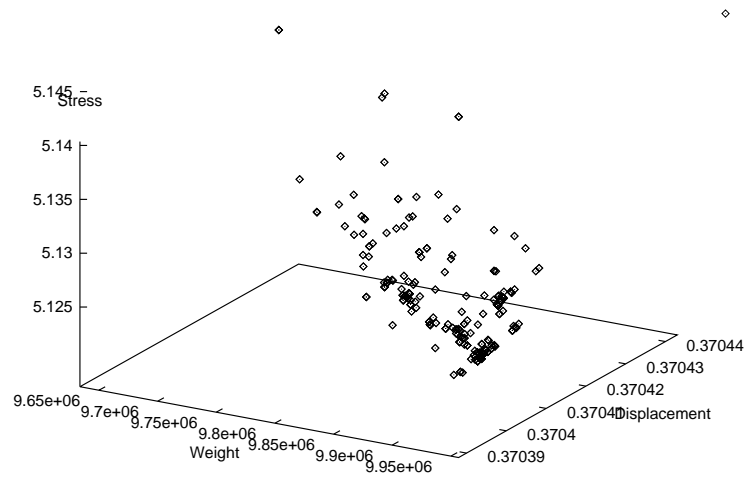


Figure 6.210: Example 6: Distribution of points using my approach based on min-max selection with sharing, using binary representation at generation 100.

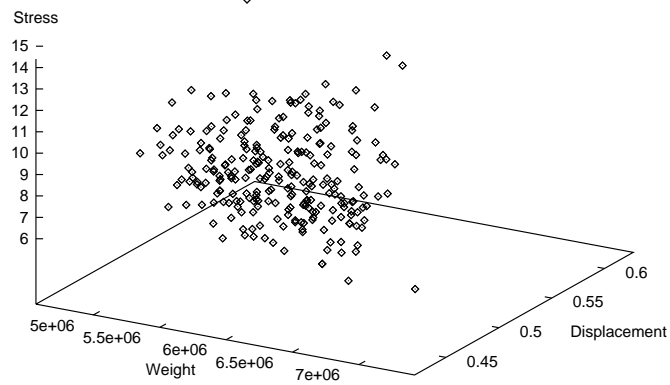


Figure 6.211: Example 6: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 20.



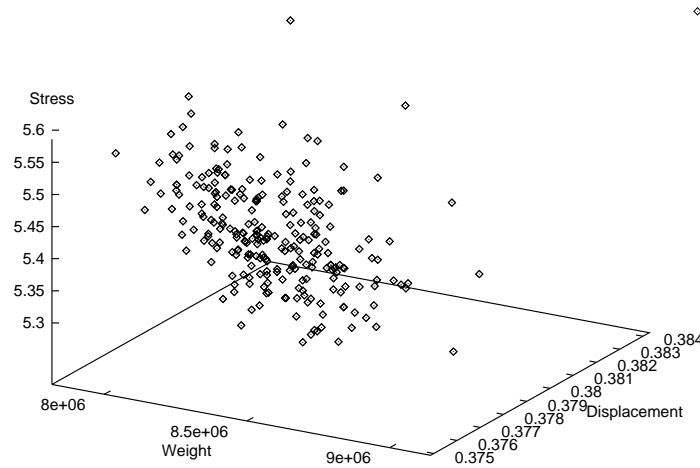


Figure 6.212: Example 6: Distribution of points using my approach based on min-max selection with sharing, using floating point representation at generation 100.

and total convergence seems imminent. The reason for this poor behavior is that the algorithm was able to find two elements of the ideal vector (displacement and stress) which are non-conflicting, and moved towards the regions where they reside regardless of the extremely high values found for the weight. That is the reason why this method reported the poorest overall solution of all the techniques under study (see Tables 6.20, 6.21, 6.22 and 6.23). Floating point representation provides a better performance in terms of population distribution, and after 20 generations the Pareto front looks more uniform (see Figure 6.211). The compromise in this case is, however, similar than before, with compromises that favor displacement and stress over weight. After 100 generations the population still looks somewhat sparse, but convergence towards the same kind of solutions found using binary representation seems unavoidable (see Figure 6.212). Floating point representation provides a slightly better overall solution, but still extremely poor when compared to the results produced by the other techniques. The value of  $\sigma_{share}$  used for solving this problem was 0.5 and the initial population used was

Method	$x_1$	$x_2$	$f_1$	$f_2$	$f_3$	$f_4$
Monte Carlo 1	0.19247	0.19511	<b>103.11</b>	39.30	708.72	236.49
Monte Carlo 1	0.02417	0.09112	214.30	<b>29.85</b>	795.79	434.79
Monte Carlo 1	0.04311	0.10200	135.1197	41.1169	<b>385.88</b>	205.76
Monte Carlo 1	0.01713	0.12098	127.45	41.90	658.27	<b>194.94</b>
Min-Max (OS)	0.19247	0.19511	<b>103.11</b>	39.30	708.72	236.49
Min-Max (OS)	0.02417	0.09112	214.30	<b>29.85</b>	795.79	434.79
Min-Max (OS)	0.04311	0.10200	135.12	41.12	<b>385.88</b>	205.76
Min-Max (OS)	0.01713	0.12098	127.45	41.90	658.27	<b>194.94</b>
GA (Binary)	0.1565	0.2000	<b>92.82</b>	41.25	679.78	201.59
GA (Binary)	0.1621	0.0963	194.43	<b>29.60</b>	805.76	444.92
GA (Binary)	0.2000	0.1972	132.20	41.94	<b>373.85</b>	194.52
GA (Binary)	0.1419	0.0965	130.06	41.93	389.06	<b>194.61</b>
GA (FP)	0.1557	0.2000	<b>91.99</b>	40.29	684.44	211.95
GA (FP)	0.2000	0.0930	168.03	<b>29.59</b>	890.92	443.34
GA (FP)	0.2000	0.2000	132.21	41.94	<b>373.83</b>	194.52
GA (FP)	0.2000	0.0096	105.22	41.94	692.73	<b>194.52</b>
Literature	0.199	0.199	<b>112.75</b>	39.06	750.60	303.09
Literature	0.175	0.114	216.76	<b>30.21</b>	713.05	452.31
Literature	0.198	0.140	133.11	41.94	<b>374.82</b>	195.23
Literature	0.191	0.198	111.99	41.94	485.66	<b>195.21</b>

Table 6.30: (Part I) Comparison of results computing the ideal vector of example 7 from Chapter 5 (design of a robot arm). For each method the best results for optimum  $f_1$ ,  $f_2$ ,  $f_3$  and  $f_4$  are shown in **boldface**. OS stands for Osyczka's Multiobjective Optimization System. (Continued in Table 6.31)

that shown in Figure 6.206, in which weight has priority over stress and displacement. The main weakness of this technique is again made evident: it tends to fail when it is possible to find solutions that produce more than one element of the ideal vector and highly disfavor the remaining objectives.

## 6.8 Example 7 : Design of a Robot Arm

In this example I will not show any graphical representation of the Pareto front, because there are too many objective functions, and plotting them is not possible anymore. So, for this problem I will concentrate on how good do all

Method	$x_3$	$x_4$
Monte Carlo 1	34.9006	2.4059
Monte Carlo 1	18.9996	14.5462
Monte Carlo 1	0.00755	0.70932
Monte Carlo 1	29.5132	0.02916
Min-Max (OS)	34.9006	2.40593
Min-Max (OS)	18.9996	14.5462
Min-Max (OS)	0.07550	0.70932
Min-Max (OS)	29.5132	0.02916
GA (Binary)	35.0000	0.4095
GA (Binary)	22.4752	15.00
GA (Binary)	0.00340	0.0000
GA (Binary)	1.8294	0.0066
GA (FP)	35.00	0.9869
GA (FP)	35.00	15.00
GA (FP)	0.0000	0.0004
GA (FP)	35.00	0.0010
Literature	34.98	5.77
Literature	10.24	14.86
Literature	0.001	0.002
Literature	14.30	0.001

Table 6.31: (Part II) Comparison of results computing the ideal vector of example 7 from Chapter 5 (design of a robot arm). OS stands for Osyczka's Multiobjective Optimization System.

the techniques in terms of getting the best overall result. First, I will start by showing the ideal vector, according to each one of the techniques used in the previous example. Again, the GA with floating point representation found the entire ideal vector. As can be seen from the results presented in Table 6.30, I found even better results than those reported in the literature.

Since I am only measuring the performance of each technique in terms of the best solution overall, there is not much to say about the results presented in Table 6.32, because they are self-explanatory. One of the most remarkable things observed when performing the experiments corresponding to this problem was that floating point representation consistently gave a poorer result with all GA-based methods. The reason for this behavior is that the solutions to this problem are very sensitive to the values of the design variables. Since floating point representation provides with shorter chromosomes, crossover tends to affect more the phenotypes in problems with a small number of variables like this. For this problem, binary representation requires strings of length 59, whereas floating point representation requires strings of only 20 genes, that being the main reason for the poorer performance of the latter, since it turns out to be harder for this scheme representation to keep good solutions through generations, unless an elitist selection strategy is implemented. This shows how in problems with short strings, a floating point representation might not work very well unless some additional restrictions are imposed when mating chromosomes.

As before, my method based on a weighted min-max approach provided the best overall result, with only a slight difference between binary and floating point representation. VEGA, Hajela's approach and my Pareto-GA method provide very good results when binary representation is used, but produce very poor solutions with floating point representation. However, some techniques such as

Method	$x_1$	$f_1$	$f_2$	$f_3$	$f_4$	$L_p(f)$
Ideal Vector		91.99	29.59	373.83	194.52	0.000000
Monte Carlo 1	0.18738	117.68	39.74	505.73	221.24	1.112581
Monte Carlo 2	0.12276	123.43	41.41	458.17	200.12	0.995563
Min-max (OS)	0.12438	135.62	35.06	740.01	306.76	2.215680
GCM (OS)	0.12438	135.62	35.06	740.01	306.76	2.215680
WMM (OS)	0.12438	135.62	35.06	740.01	306.76	2.215680
PMM (OS)	0.12438	135.62	35.06	740.01	306.76	2.215680
NMM (OS)	0.12438	135.62	35.06	740.01	306.76	2.215680
GALC (B)	0.20000	102.18	41.94	529.01	194.52	0.943299
GALC (FP)	0.20000	161.89	30.85	878.20	430.70	3.365909
Lexicographic (B)	0.04690	129.19	41.29	414.91	202.44	0.950512
Lexicographic (FP)	0.20000	105.00	41.74	694.10	197.74	1.425351
VEGA (B)	0.18010	129.71	41.90	393.13	194.90	0.879695
VEGA (FP)	0.20000	105.20	41.94	692.83	194.79	1.415545
NSGA (B)	0.13680	115.61	41.69	465.15	197.78	0.926816
NSGA (FP)	0.14100	93.16	41.48	676.66	199.22	1.248742
MOGA (B)	0.17370	144.92	34.49	637.22	316.49	2.072441
MOGA (FP)	0.20000	165.01	29.73	885.55	438.00	3.418972
NPGA (B)	0.08520	109.84	41.51	695.58	204.99	1.511325
NPGA (FP)	0.20000	227.94	29.75	631.28	449.13	3.480962
Hajela (B)	0.19990	132.75	41.75	376.58	196.89	0.873619
Hajela (FP)	0.20000	166.52	47.46	841.71	394.29	3.692761
GAminmax1 (B)	0.16430	132.20	41.94	373.87	194.52	0.854600
GAminmax1 (FP)	0.20000	132.19	41.94	373.93	194.52	0.854665
GAminmax2 (B)	0.18930	124.37	41.86	413.73	195.25	0.877150
GAminmax2 (FP)	0.20000	200.75	31.01	686.86	430.04	3.278562

Table 6.32: (Part I) Comparison of the best overall solution found by each one of the methods included in MOSES for the seventh example (design of a robot arm). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective). (Continued in Table 6.33)

Method	$x_2$	$x_3$	$x_4$
Monte Carlo 1	0.18074	13.47721	1.5555
Monte Carlo 2	0.17042	9.26852	0.33446
Min-max (OS)	0.09609	29.9961	6.9961
GCM (OS)	0.09609	29.9961	6.9961
WMM (OS)	0.09609	29.9961	6.9961
PMM (OS)	0.09609	29.9961	6.9961
NMM (OS)	0.09609	29.9961	6.9961
GALC (B)	0.20000	20.2738	0.0132
GALC (FP)	0.06450	35.0000	15.000
Lexicographic (B)	0.01970	10.5668	0.4671
Lexicographic (FP)	0.02720	35.0000	0.8189
VEGA (B)	0.08990	15.3888	0.4392
VEGA (FP)	0.01320	35.0000	0.0188
NSGA (B)	0.06090	20.9519	0.4369
NSGA (FP)	0.20000	35.0000	0.2221
MOGA (B)	0.00630	35.0000	9.1130
MOGA (FP)	0.08090	35.0000	15.000
NPGA (B)	0.04760	35.0000	0.6984
NPGA (FP)	0.20000	0.0009	15.000
Hajela (B)	0.11240	0.0391	0.1537
Hajela (FP)	0.20000	35.000	10.0624
GAminmax1 (B)	0.20000	0.0059	0.0006
GAminmax1 (FP)	0.05000	0.0122	15.000
GAminmax2 (B)	0.15480	21.6953	0.24970
GAminmax2 (FP)	0.06390	35.0000	15.000

Table 6.33: (Part II) Comparison of the best overall solution found by each one of the methods included in MOSES for the seventh example (design of a robot arm). GA-based methods were tried with binary (B) and floating point (FP) representations. The following abbreviations were used: OS = Osyczka's System, GCM = Global Criterion Method (exponent=2.0), WMM (Weighting Min-max), PWM (Pure Weighting Method), NWM (Normalized Weighting Method), GALC = Genetic Algorithm with a linear combination of objectives using scaling. In all cases, weights were assumed equal to 0.25 (equal weight for every objective).

Hajela's method, prematurely converged to the solution presented in Table 6.32. In these tests,  $\sigma_{share} = 0.1$  for all Pareto-based approaches, except mine, which uses  $\sigma_{share} = 0.5$ .

## 6.9 Example 8 : Design of a Combinatorial Circuit

This is by far, the hardest problem faced by the GA-based methods analyzed in this thesis. To solve it, many obstacles had to be faced. First, the two objectives indicated in Chapter 4 (i.e., functionality and the minimization of the number of gates other than WIRE) had to be combined to get the best performance of the GA, which means that the problem had to be considered as a single-objective optimization problem. The GA is able to solve this single-objective optimization problem when a proper population size and parameters were provided. In that sense, I was unable to reproduce the results presented by Louis in his dissertation [204] for the traditional GA. The fitness function that he suggests did not work at all, and I was unable to achieve convergence using the genetic algorithm with different parameters. In this respect, I should point out that the convergence graph that Louis shows on page 67 of his dissertation [204] indicates that the maximum fitness that he could get after 50 generations was around 41, which is below the bound of feasible solutions (a feasible solution has a fitness of 44). In my experiments, I had to devise a different fitness function, I used a fairly large population size (500), and a combination of low crossover and high mutation rate (for example, crossover rate = 0.1 and mutation rate = 0.4) to find feasible solutions. This clearly indicates the difficulties of the GA to keep highly fit schemas, since crossover will tend to disrupt good solutions. This makes

this problem very challenging and interesting to apply the GA. Also, because of its encoding it is not suitable for traditional mathematical programming techniques.

In the analysis of this problem, I will show the fitness function that was found most appropriate for the traditional GA, together with convergence graphs that will show the efficiency of the technique. On the other hand, it is also interesting to look at the physical representation of the solutions found by the GA, and compare them with those previously reported in the literature [205].

First of all, the main problem faced when trying to design a 2-bit adder with a GA was to devise a good fitness function. The initial approach was to use a slight variation of the function suggested by Louis in his dissertation [204], which consists of the number of correct additions performed. Instead of checking on the 3 possible outcomes for each of the 16 possible combinations of inputs, I checked only on the final result on each case, requiring, therefore, only 16 comparisons instead of the 48 originally required. This fitness function worked fine, assuming that proper parameters were provided for the GA. In the results that follow, I used a population size of 500 chromosomes, a crossover rate of 0.1, a mutation rate of 0.4, and I ran the GA for 100 generations. I also found more suitable to incorporate the second objective as a reward to the fitness function, since none of the multiobjective optimization techniques implemented in MOSES was able to handle such value as a completely separate objective function. In fact, the fitness function that I used turns out to be a linear combination of the two objectives (i.e., an addition of them). I ran a loop over the 16 different types of gates employed for each design, and whenever a WIRE was employed, the fitness function was rewarded with a bonus of unitary value. Therefore, the best solutions found use 9 WIREs, since their fitness is 25 (a functional circuit has a fitness of 16, and since the fitness is increased by one for each WIRE that uses, a fitness of 25



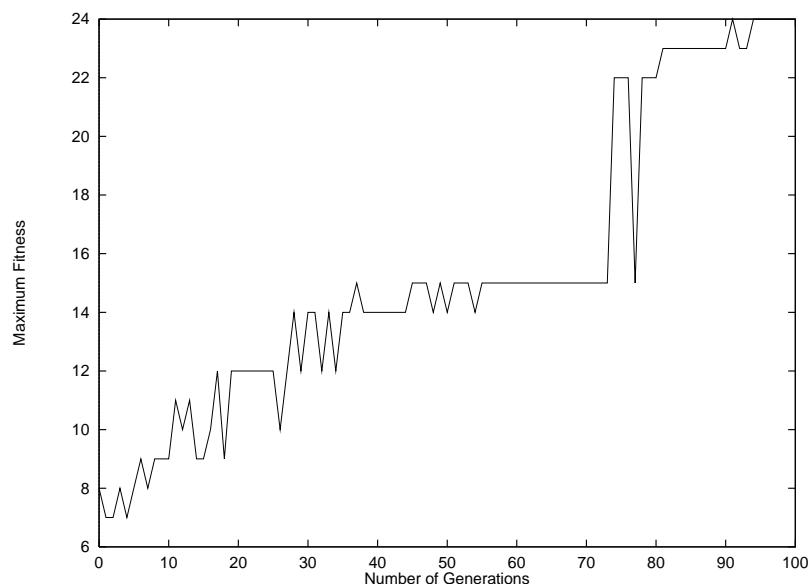


Figure 6.213: Convergence history of the GA solving problem No. 8 (binary representation). The evolution of the maximum fitness is displayed through 100 generations.

means that we have a functional circuit that uses 9 WIRES). That corresponds to a better solution than those provided by Louis (his sample solution produced with the traditional GA has a fitness of 24, and his sample solution using his masked crossover operator has a fitness of 21). Notice that since I did not use any elitist selection strategy in my tests (I did not want to alter the basic structure of MOSES with the details of this problem), the graphs will not show a monotonic increment of the fitness values.

The convergence of the GA using a linear combination of the two objectives is shown in Figures 6.213 and 6.214. The first graph shows the evolution of the maximum fitness, and the second shows the evolution of the average fitness. We can see how the GA requires over 70 generations to come up with feasible solutions, but once it finds one, it can easily make the population converge towards them. It is interesting to notice that, due to the nature of the problem, the GA is able to generate several different solutions even without using any kind of sharing technique. Notice that only binary representation was used, since the problem

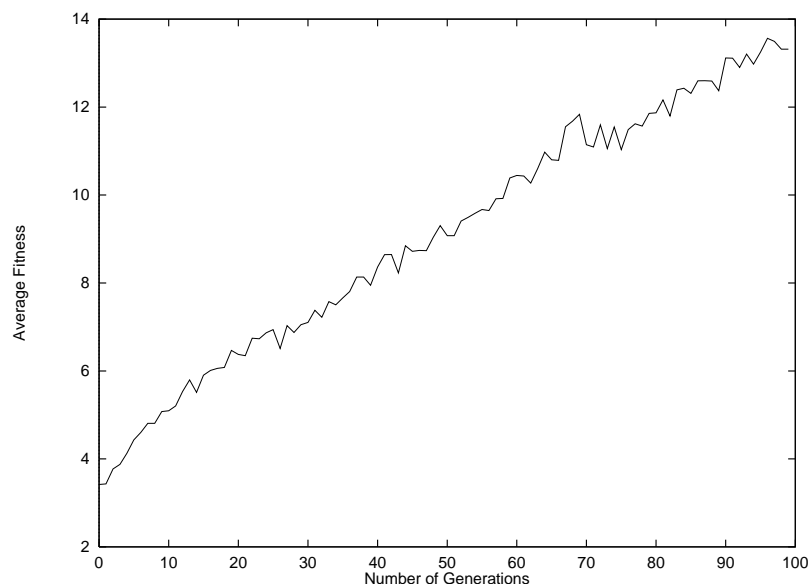


Figure 6.214: Convergence history of the GA solving problem No. 8 (binary representation). The evolution of the average fitness is displayed through 100 generations.

is too sensitive to the scheme representation to allow the use of floating point representation. The fact that floating point representation does not seem to work with this problem should not surprise us, since it is more suitable for numerical optimization problems and in this case the concept of design is taken from a more creative perspective, with a search space shaped in a form in which binary representation seems the “natural” and obvious encoding.

Finally, since I considered interesting to analyze some of the solutions generated by the GA, I decided to include some of the circuits corresponding to some of the designs produced by the GA. Figures 6.215 and 6.216, show some examples of such designs. The solution provided by Louis in his dissertation [204] is shown in Figure 6.217 (this is the solution corresponding to the use of a traditional GA with an elitist selection strategy). As can be observed in these pictures, my designs use an extra WIRE, which means that they require one gate less than the design produced by Louis. According to my own goals set for this

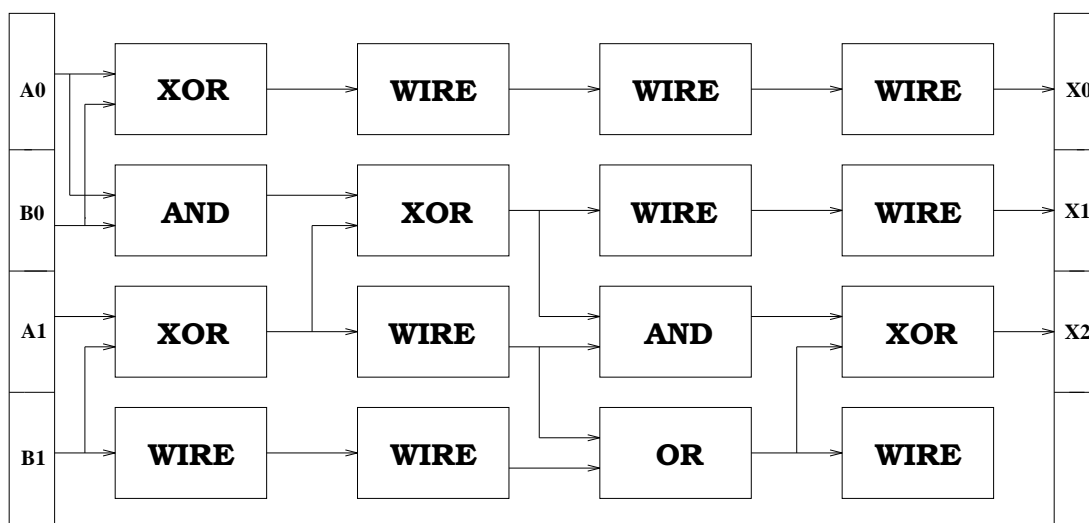


Figure 6.215: Example 8: A sample circuit design produced by MOSES using binary representation.

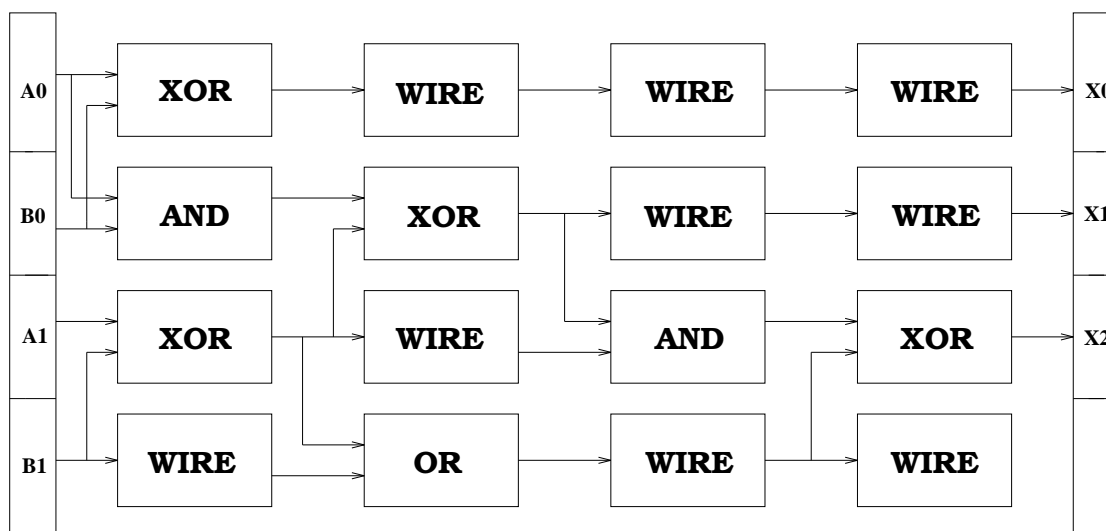


Figure 6.216: Example 8: A sample circuit design produced by MOSES using binary representation.

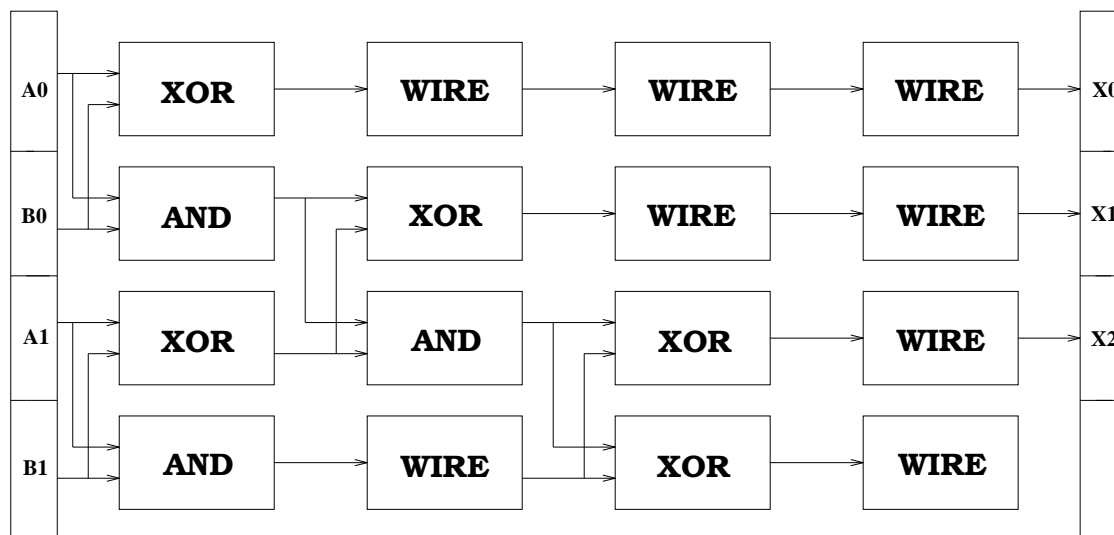


Figure 6.217: Example 8: A sample circuit design produced by the classical genetic algorithm with an elitist selection strategy used by Sushil Louis.

problem, my designs are better than Louis' designs, because they are functional and maximize the number of WIREs within the circuit.

## 6.10 Critique of each Method

After analyzing all the results previously presented, I have derived some conclusions about the behavior of each technique that will be summarized in the following section, indicating the advantages and disadvantages of each technique, and where seems most appropriate to use them.

### 6.10.1 Monte Carlo Method 1

- Fast convergence.
- Reasonable good solutions, but only one at a time.
- It can not generate the Pareto front.

- It computes the best overall solution using the min-max method, but without explicitly asking the ideal vector.
- It can handle multiple objectives and multiple constraints, but if there are too many variables (more than 4) and/or their ranges are too large, and therefore the search space is too large to track, then this random method is not very effective.
- This method is less efficient (computationally speaking) than Monte Carlo Method 2, because it explores the space twice, but it uses less memory.

### 6.10.2 Monte Carlo Method 2

- Very fast convergence.
- It is very efficient because the space of variables is explored only once, but it requires much more computer memory since the whole set of Pareto optimal solutions has to be stored.
- It does not generate the Pareto front, but it only outputs the best non-dominated solution.
- It is more efficient when there are many constraints than Method 1, since the Pareto region is smaller.
- Monte Carlo methods in general are very suitable for generating a rough solution when the size of the search space is not too large, and when time is an issue, since they are very fast at that. However, when the Pareto set is too large or there are many variables involved, these methods turn out to be very inefficient and will have difficulty finding a reasonable solution.

### 6.10.3 Osyczka's Multiobjective Optimization System

- Relatively fast convergence, but slower than Monte Carlo methods.
- It may find reasonable good solutions but only if we give a very close approximation. If we provide a guessing point too far from the global optimum, the single optimization technique used to compute the ideal vector could take a long time to center the proper portion of the search space or even fail to do so.
- It can generate a very small fraction of the Pareto front (most of the time only one point), and only if a sufficient amount of points is used by the method (normally a large number).
- It computes the best overall solution using 5 different methods, and the user is given the choice of providing the ideal vector or to let the system compute it, and it offers the possibility of being used as an interactive system.
- It can handle multiple objectives and multiple constraints, and also several single-criterion optimization may be incorporated (only the flexible tolerance method is incorporated in my implementation). Its only drawback in that sense is the amount of memory available, because it could require very large arrays when there are several variables (with large ranges) involved.
- The system can not deal with discrete variables, since it assumes that every variable is continuous.
- Although many different options are available to the user in this system, it was designed to be used in an interactive manner, which means that the user has to provide a lot of information and try it over and over again.

#### 6.10.4 GA-based Linear Combination of Objectives

- It is very easy to implement, and as efficient as the traditional SGA.
- It requires certain knowledge of the problem, since scaling is necessary to avoid premature convergence problems. In most cases, it will, anyway, converge towards a single solution.
- It is very efficient (i.e., it is fast and it can provide very good overall results) if proper scaling factors are used.
- It is recommended as an exploratory tool together with some of the Monte Carlo techniques previously described.

#### 6.10.5 Lexicographic Method

- It is very easy to implement, because the traditional GA only needs to be extended using a random flag to choose among the different objective functions.
- It is the fastest GA-based technique, since no additional computations need to be done, the original complexity of the traditional GA is kept.
- It does not work very well when the number of objective functions increases.
- It works surprisingly well with small problems, and is able to draw the contour of the Pareto front very accurately. However, in larger problems is not very good at generating the Pareto front.
- Randomly selecting an objective is equivalent to averaging fitness across fitness components, each component being weighted by the probability of each objective being chosen to decide each tournament [188]. However, since

pairwise comparisons are used, this method is essentially different from a linear combination of objectives, because scale information is ignored.

### 6.10.6 VEGA

- It is very efficient and easy to implement and understand.
- It is not appropriate for problems with a concave trade-off surface, because it will not be able to find it.
- It is able to generate a good portion of the Pareto front in most cases, but it tends to converge to a single solution if either too many generations or floating point representation is used, since the problem of “speciation” becomes critical.
- It is not recommended to use it with a floating point representation scheme, unless only a rough approximation of the best trade-off is desired, but it will not do better than a linear combination of objectives and it takes about the same computation time.

### 6.10.7 NSGA

- The selection strategy is somewhat slow, because checking for non-dominance requires an  $O(n^2)$  algorithm, and the computation of the niche counts requires also  $O(n^2)$  operations, where  $n$  refers to the population size.
- Not very good for generating the best overall solution, because the real objective function values are never used by the algorithm.
- It can generate good Pareto fronts some times, but it can not keep them for too long (no more than 100 generations).



- Sharing is done in the parameter values instead of the objective values, to ensure a better distribution of individuals, and to let multiple equivalent solutions exist.
- It does not consider the inclusion of constraints, so they have to be considered independently by the user, and that could imply an increase in the complexity of the algorithm.
- The value of  $\sigma_{share}$  has to be estimated beforehand, because that could affect, under certain conditions, the performance of the algorithm.
- Floating point representation is not recommended unless very few generations are used.
- The use of stochastic remainder selection seriously degrades the overall performance of the algorithm.

### 6.10.8 MOGA

- It is faster than NSGA, because its ranking algorithm is more efficient and it uses a tournament selection strategy.
- Again, how to choose the value of  $\sigma_{share}$  is an issue, even when sometimes there is no significant change in the results when this parameter is modified.
- Sharing is done on the objective value space, which means that two different vectors with the same objective function values can not exist simultaneously in the population under this scheme. This is apparently undesirable, because these are precisely the kind of solutions that the user normally wants, but the method works surprisingly well in practice.

- It can display a good portion of the Pareto front when it is used with a binary encoding for several generations. It does not converge towards a single solution if appropriate sharing parameters are chosen, but it is better if floating point representation is used because it displays a clearer Pareto front and it requires less generations.
- A reasonably good overall result can be found with this technique, because the fitness values are used to guide the search, even when the niching mechanism avoids the technique to find the best overall result.
- It is very stable if sharing parameters are chosen appropriately, but it could produce fronts that contain an excessive amount of dominated points even after many generations. In some cases, it seems to be more capable to display the whole feasible region, instead of just displaying the Pareto front.
- The algorithm does not check feasibility of the solutions encoded in the individuals, so that there could be convergence towards infeasible solutions.

#### 6.10.9 NPGA

- This is the fastest of the Pareto-ranking techniques, if the tournament size for determining non-dominance ( $t_{dom}$ ) is not very large (which is normally the case).
- It produces very clear Pareto fronts, and it keeps the points that form the curve even after many generations.
- It produces the largest section of the Pareto front of all Pareto-ranking based techniques.
- It does not check feasibility of the points generated.

- Besides requiring the value of  $\sigma_{share}$  for forming niches, it requires the size of the tournament for determining non-dominance ( $t_{dom}$ ), and the performance of the techniques highly relies on such values.
- Floating point representation produces good approximations of the Pareto front, and presents a remarkable robustness after many generations, although sometimes it does not generate the non-dominated points where the global optimum reside.
- When using floating point representation, the technique becomes more sensitive to the values of its parameters, and a slight change of them abruptly modifies the results, and premature convergence is the normal trend of the method.
- Using binary representation, good overall solutions may be obtained, but not better than those generated by simple linear combination of objectives or by the Lexicographic method.

#### 6.10.10 Hajela's Method

- This is a non-Pareto approach based on the weighted min-max method.
- The determination of the weights could be an issue, although the designer normally knows what weights to assign.
- It uses the ideal vector, but if it is not known, the method can still be used providing a utility function (i.e., an estimation of what would be the desirable solution).
- It works very well at early stages of the search, generating a very clear portion of the Pareto front and an excellent best overall solution, but it

quickly converges towards a single non-dominated solution after a few more generations even if large populations are used.

- It uses sharing in the objective function space, and applies mating restriction to try to preserve the best individuals, requiring therefore the determination of  $\sigma_{share}$  and the mating restriction threshold to operate well.
- The method is very sensitive to changes in the values of its parameters.
- The use of floating point representation produces premature convergence of the GA.
- The method does not check feasibility of the solutions encoded, and sometimes converges towards invalid solutions.
- It only produces reasonable good trade-offs when used with binary representation during a few generations. Otherwise, it has a very poor performance.

#### 6.10.11 GA with a Weighted Min-Max Technique

- This method is based on a weighted min-max optimization approach.
- It generates only points within the feasible region, and enforces feasibility in the crossover and mutation operators across generations.
- It requires the determination of a set of weights for each one of the objective functions.
- It requires the ideal vector, but it may also be used with a utility function.
- This method is intended to be used with a distributed or a parallel GA, since it transforms the multiobjective optimization problem into  $k$  single-optimization problems where  $k$  is the number of objective functions.

- If run on a sequential machine, this method is really slow compared to the previous methods.
- It generates an excellent contour of the Pareto front with very few non-dominated points evenly distributed.
- It generates the best overall solution of all methods.
- It is very robust since each objective is treated separately.
- It does not require sharing or any other parameters.
- It works almost equally well with both binary and floating point representation. However, with floating point representation is able to find the best overall solutions.

#### 6.10.12 GA with Min-Max Binary Tournament Selection and Sharing

- It uses a min-max binary tournament selection scheme (deterministic).
- The ideal vector is determined for each population and non-dominance is measured with respect to such vector in each generation.
- It uses sharing to decide ties in a similar manner as in NPGA, counting the niche sizes of the two competitors in terms of their objective function values (the one with less individuals in its niche wins).
- It is extremely fast, since it does not require any sort of ranking or sorting of the population.
- It is sensitive to changes in the value of  $\sigma_{share}$ , and it normally requires higher values than those used by other Pareto-based approaches.

- It finds a large portion of the Pareto front in a reasonable number of generations, and it keeps most of it after many generations.
- It produces a very good overall solution using binary representation.
- It uses the objective function values, but it never combines those functions in any way.
- It does not require large populations to produce reasonable approximations of the Pareto front.
- It tends to converge to unique (and usually poor) solutions when there is at least one achievable element from the ideal vector.

## Chapter 7

### Discussion

Some of my experiences regarding population policies and sharing parameters are discussed in this chapter, in the hope of clarifying the issues of adjusting parameters in multiobjective optimization problems. The difficulties that some problems present, such as the design of combinatorial circuits, are also addressed, and a brief discussion on representation issues and the importance of choosing a good fitness function is provided. Another important point is made regarding the incorporation of knowledge from the domain into the genetic algorithm. In that respect, some of the most important work in the area is referenced, and my own ideas are discussed. Finally, an expert system is introduced that advises on the most proper technique to use (from those contained in MOSES) depending on the user's needs and the knowledge that he/she has about the characteristics of the problem. This system is intended to be used as a front end to MOSES, so that the user knows the most proper procedure to follow. Hopefully, all the information provided in this chapter will be enough for any novice user to exploit most of the potential of MOSES, even if he/she has little or no knowledge of genetic algorithms.

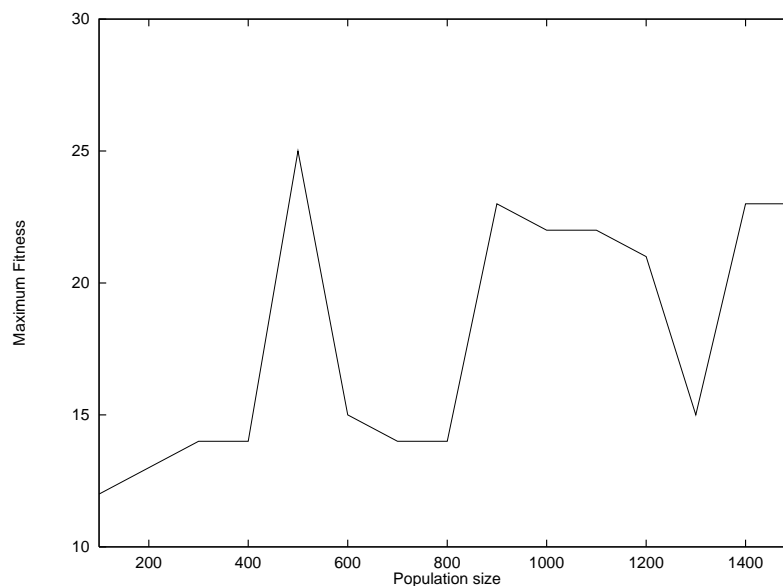


Figure 7.1: Relation between population size and maximum fitness for the example No. 8

## 7.1 Population Policies

One of the issues that normally worries GA practitioners is the fine tuning of parameters. In this thesis, I have addressed the problem of adjusting crossover and mutation rates using a sequential procedure that is equivalent to a dynamic adjustment of parameters. However, this procedure uses a constant population size and its behavior depends on this parameter. There is some important work on defining optimum population sizes for serial and parallel genetic algorithms done by Goldberg [206] [207], Reeves [208] and Alander [209]. In Goldberg's work, it is desired to maximize the number of new schemata per individual, so that the optimal population for binary-coded strings grows exponentially with the length of the string. Reeves [208] and Alander[209], on the other hand suggest that it is possible to use very small populations for numerical optimization problems, without degrading too much the performance of the GA. Alander suggests that we choose population sizes between  $L$  and  $2 \times L$ , where  $L$  is the length of the chromosome, and Reeves further suggests the use of the probability that at least



M	$L_p(f)$ (B)	$L_p(f)$ (FP)	E
50	2.836104	2.921610	5000
100	2.653560	2.607544	10000
150	2.683763	2.627390	15000
200	2.605933	2.621065	20000
250	2.599961	2.608188	25000
300	2.603364	2.616161	30000
350	2.599006	2.627159	35000
400	2.619167	2.615799	40000
450	2.581377	2.582281	45000
500	2.628888	2.568597	50000

Table 7.1: Comparison of results using different population sizes with my min-max method that uses sharing. M is the population size,  $L_p(f)$  is the maximum deviation with respect to the ideal vector and E is the number of function evaluations required. These results correspond to example 1 from Chapter 5 using binary (B) and floating point (FP) representations.

one allele is present at each locus in the initial population as the lower bound to estimate our population size. As Reeves points out, one of the main issues when using the GA is that if we have to experiment too much, or if we have to use population sizes extremely large, then the technique will not be a good competitor against other heuristics such as simulated annealing and tabu search, mainly in the domain of numerical optimization. Reeves' idea is valid, but my experiments with single and multiple-objective optimization indicate that the GA using a larger alphabet (namely a floating point representation) is able to find better solutions than binary representation, using the exact same parameters (i.e., population size, crossover and mutation rates). Also, if we move into problems in which the concept of design is taken in a more creative sense rather than from a numerical point of view, then we will see that defining population sizes can be even more difficult.

I will start my analysis by showing the effect of the population size on the performance of the GA, measured in terms of the best overall result, because what

we are interested in right now is knowing how good the GA can be at finding a single trade-off. For these experiments, I will be using my min-max method that uses sharing and some of the examples from Chapter 5. First, I will show some of the analysis corresponding to example 1 (design of an I-beam) using binary and floating point representations (see Table 7.1). We can see how when binary representation is used, there is no big difference between the results using larger populations than the one I used in my experiments ( $M=100$ ). The chromosome length in this case is 49, which means that I am using a conservative approach of requiring a population of at least  $2 \times L$ . From these results, we can see that even if we increase the population to 450 chromosomes, the result is improved by only about 3 %, while the computational cost is increased 4.5 times. The following expression given by Reeves [208] can be used to compute the minimum population size required to ensure enough diversity in the population:

$$P_2^* = (1 - (1/2)^{M-1})^L \quad (7.1)$$

where  $P_2^*$  is the probability that at least one allele is present at each locus in the initial population, using binary representation. If we want to have a probability of 100 %,  $M$  must be chosen as at least 40. However, such a small value will produce a value  $L_p(f) = 2.752742$ , which is not very good, although is better than the value produced with 50 chromosomes. Furthermore, in cases such as example 6 in which the length of the string for binary representation is very large (493 in case of example 6), his formula suggests using population sizes as small as 40, when in practice even a population of 500 was unable to find the ideal vector.

Floating point representation requires chromosomes of length 16, and it produces better results than binary representation in average, and it generates the best result overall. For this example, a value of  $\sigma_{share} = 0.5$  was used, and the

M	$L_p(f)$ (B)	$L_p(f)$ (FP)	E
50	0.351841	0.316188	5000
100	0.049133	0.096331	10000
150	0.163010	0.004151	15000
200	0.154574	0.000812	20000
250	0.117164	0.004070	25000
300	0.031906	0.012259	30000
350	0.189382	0.012264	35000
400	0.156802	0.000029	40000
450	0.175122	0.000029	45000
500	0.141048	0.000003	50000

Table 7.2: Comparison of results using different population sizes with my min-max method that uses sharing. M is the population size,  $L_p(f)$  is the maximum deviation with respect to the ideal vector and E is the number of function evaluations required. These results correspond to example 2 from Chapter 5 using binary (B) and floating point (FP) representations.

number of function evaluations was calculated by multiplying  $M \times k \times g$ , where  $k$  is the number of objectives and  $g$  is the number of generations. Even when the alphabet used by floating point representation is considerably larger, we can see how the results are consistently better. The same behavior is exhibited when we perform single-objective optimization, as we saw in Chapter 6. This contradicts Reeves' theory according to which a higher alphabet such as this would require considerably larger populations to converge. My experiments show exactly the opposite, since I always found it possible to generate better solutions (and in less time) using floating point representation.

Another study of the effect of population size was conducted on example 2 (machining recommendations) from Chapter 5. The results are shown in Table 7.2. Again, floating point representation is able to surpass binary representation on average, and it finds a result that practically matches the ideal vector (its deviation is almost zero). The chromosome length in this problem is 40 for binary

M	$L_p(f)$ (B)	$L_p(f)$ (FP)	E
50	2.054163	3.651128	5000
100	2.214113	1.677212	10000
150	2.021498	2.269683	15000
200	1.740228	1.706499	20000
250	1.864798	1.827804	25000
300	1.731681	1.961402	30000
350	1.754297	1.524895	35000
400	1.780415	1.582077	40000
450	1.849722	1.643386	45000
500	1.816256	1.624714	50000

Table 7.3: Comparison of results using different population sizes with my min-max method that uses sharing. M is the population size,  $L_p(f)$  is the maximum deviation with respect to the ideal vector and E is the number of function evaluations required. These results correspond to example 3 from Chapter 5 using binary (B) and floating point (FP) representations.

representation and 13 for floating point representation. One interesting observation with regards to this problem is that the sharing factor used was different. I used  $\sigma_{share} = 1.0$  for binary representation and  $\sigma_{share} = 0.001$  for floating point representation. This factor also plays a crucial role in terms of convergence and population diversity. For example, I found in this problem that a low value of  $\sigma_{share}$  with floating point representation gives a much better overall result, but makes the population converge to a unique solution most of the time. However, if a higher value is used (for example  $\sigma_{share} = 0.01$ ) there is a better distribution of the population, although the best overall result turns out to be poorer. Using binary representation, the converse seems to be true. An interesting aspect of the importance of  $\sigma_{share}$  is that it can be used to switch the GA from a mathematical programming tool in which we aim to find only one trade-off (the best available) to a Pareto ranking technique in which we want to find the entire Pareto front. More in this respect will be discussed in the following section.

The last analysis that I did was on example 3 from Chapter 5. The results are presented in Table 7.3. When binary representation is used in this example, a chromosome length of 31 is required, and when floating point representation is used, the length goes down to only 10. For these experiments, I used a value of  $\sigma_{share} = 0.5$  both for binary and floating point representation, but any change to this value did not seem to affect the overall result. It should be noticed that even when the average result is better for binary representation in this case, the best overall solution was found with floating point representation, and it produces results not too different to those produced with binary representation except for the first case (when  $M=50$ ) which is the source of discrepancy in the performance of these two representation schemes.

My experience in numerical optimization has shown me that normally a rate of at least 1:1 between population size and chromosome length produces reasonable results, and a rate of 2:1 is advised whenever possible (i.e., when the computational cost is not too high). Nevertheless, there are always exceptions to this rule, and in some problems very sensitive to the representation scheme used, even a slight change in the population size produces a significant difference in the results. To illustrate my point, I am going to use example 8 from Chapter 5 (design of a combinatorial circuit). I tried to solve this problem using different population sizes while keeping constant the remaining parameters of the GA. The results are shown in Figure 7.1, and indicate an erratic behavior of the GA. As can be seen in this table, the GA is able to find the optimum result only when a population of 500 chromosomes is used. Interestingly, it does not find the solution even with larger populations as one would expect, and in several cases it even fails to find a feasible result. The logic behind this behavior is that this problem is very sensitive to the use of operators, and because of the constraints imposed on

the inputs and the possible gates to be employed in the design, some populations disfavor the propagation of the good solutions and get stuck in a non-feasible result. Also, the extremely high mutation rates (0.4 in this case) make the GA to behave more like a hill-climbing technique than like a more robust evolutionary approach. It is interesting to point out that even when the design of circuits is considered to be more within the domain of genetic programming, the traditional GA is perfectly capable to handle it if a proper representation is used and if a good technique for adjusting its parameters is employed. For instance, in example 8 from Chapter 5, I encoded the gates in a column order (in a top-down manner), because it reflects better the interactions among inputs and outputs. However, as Louis indicates [205], a row order from left to right does not work at all since the problem becomes highly deceptive and the GA can not find the proper building blocks to ensure convergence towards the proper areas of the search space. Also, the use of a floating point representation did not work, regardless of the parameters used including population size. It would be interesting however, to experiment with some of the crossover approaches that have been proposed in the literature such as those mentioned in Eshelman and Schaffer's work [161].

## 7.2 Niching Parameters

My experiments with fitness sharing for each objective within my min-max strategy gave promising results at the beginning, but it had some problems with the last 4 problems, both in terms of finding the best overall result and in terms of providing a good distribution of the population. The value of  $\sigma_{share}$  plays a key role in the performance of the algorithm, and therefore, I decided to run some tests with different values to see its effect on the solutions found. My results are summarized in Tables 7.4, 7.5 and 7.6 for examples 1, 2 and 3 respectively. For

$\sigma_{share}$	$L_p(f)$ (B)	$L_p(f)$ (FP)
0.0	2.628584	2.570911
0.00001	2.660908	2.623860
0.0001	2.660908	2.623860
0.001	2.660908	2.635036
0.01	2.660908	2.635036
0.1	2.922974	2.631968
0.5	2.653560	2.607544
1.0	2.720461	2.607544
10.0	2.683472	2.575646
100.0	2.603680	2.567910

Table 7.4: Comparison of results using different values of  $\sigma_{share}$  with my min-max method that uses sharing.  $L_p(f)$  is the maximum deviation with respect to the ideal vector. These results correspond to example 1 from Chapter 5 using binary (B) and floating point (FP) representations.

$\sigma_{share}$	$L_p(f)$ (B)	$L_p(f)$ (FP)
0.0	0.215196	0.066086
0.00001	0.033297	0.096233 (*)
0.0001	0.020111	0.096753
0.001	0.200788	0.096331 (*)
0.01	0.325226	0.125649
0.1	0.403476 (*)	0.054456
0.5	0.295288	0.054456
1.0	0.049133	0.270773 (*)
10.0	0.173621	0.096753
100.0	0.946293	0.095485 (*)

Table 7.5: Comparison of results using different values of  $\sigma_{share}$  with my min-max method that uses sharing.  $L_p(f)$  is the maximum deviation with respect to the ideal vector. These results correspond to example 2 from Chapter 5 using binary (B) and floating point (FP) representations. The asterisk (\*) indicates total convergence of the population to a unique solution.

$\sigma_{share}$	$L_p(f)$ (B)	$L_p(f)$ (FP)
0.0	2.491438	1.828912
0.00001	2.214113	1.677212
0.0001	2.214113	1.677212
0.001	2.214113	1.677212
0.01	2.214113	1.677212
0.1	2.214113	1.677212
0.5	2.214113	1.677212
1.0	2.214113	1.677212
10.0	2.214113	1.708526
100.0	1.673000	1.508644

Table 7.6: Comparison of results using different values of  $\sigma_{share}$  with my min-max method that uses sharing.  $L_p(f)$  is the maximum deviation with respect to the ideal vector. These results correspond to example 3 from Chapter 5 using binary (B) and floating point (FP) representations.

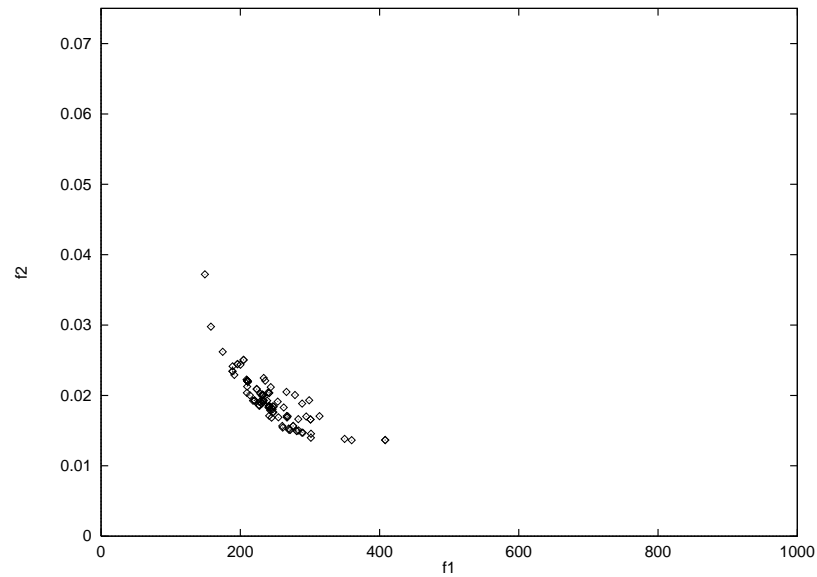


Figure 7.2: Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 0.0001 under binary representation.



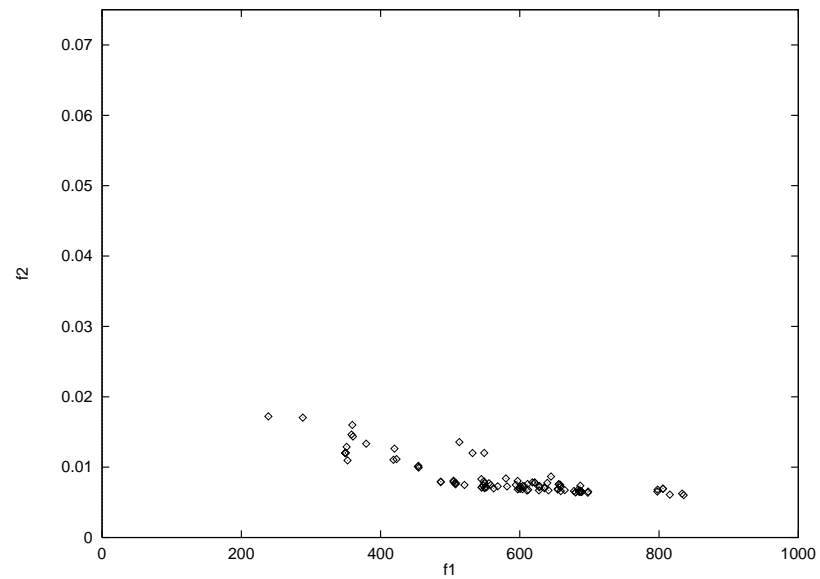


Figure 7.3: Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 0.0001 under floating point representation.

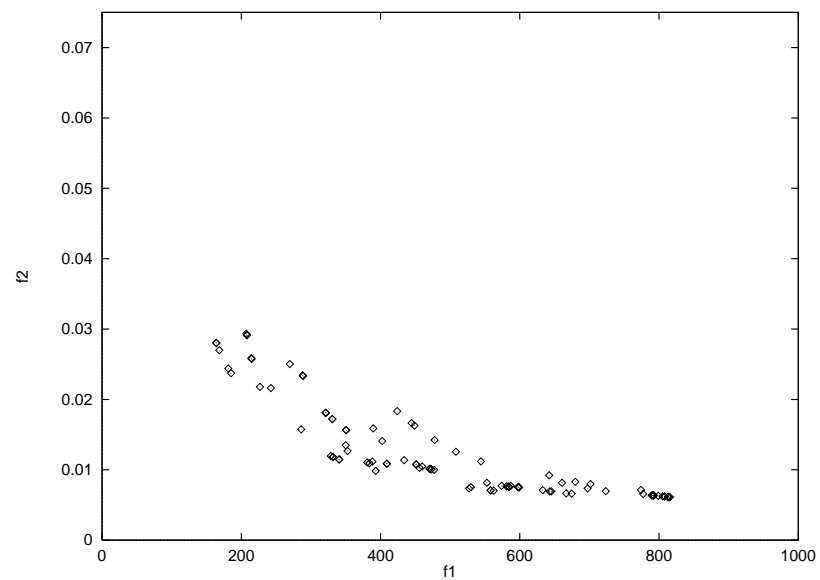


Figure 7.4: Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 100.0 under binary representation.

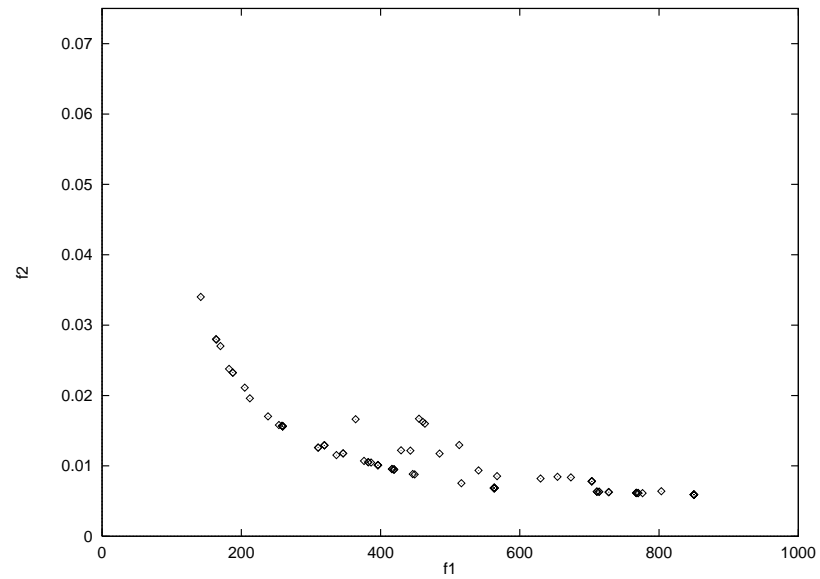


Figure 7.5: Distribution of points in the objective function domain for example 1 using my min-max strategy with a sharing factor of 100.0 under floating point representation.

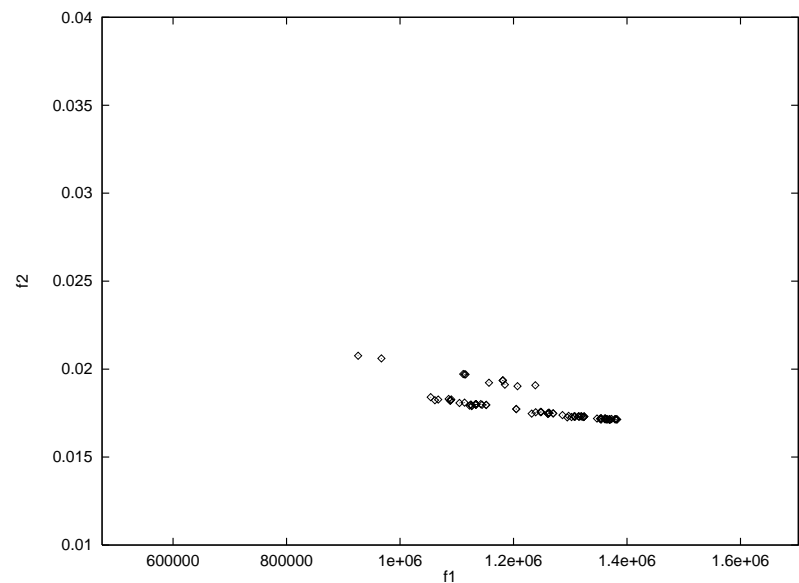


Figure 7.6: Distribution of points in the objective function domain for example 3 using my min-max strategy with a sharing factor of 0.0001 under binary representation.

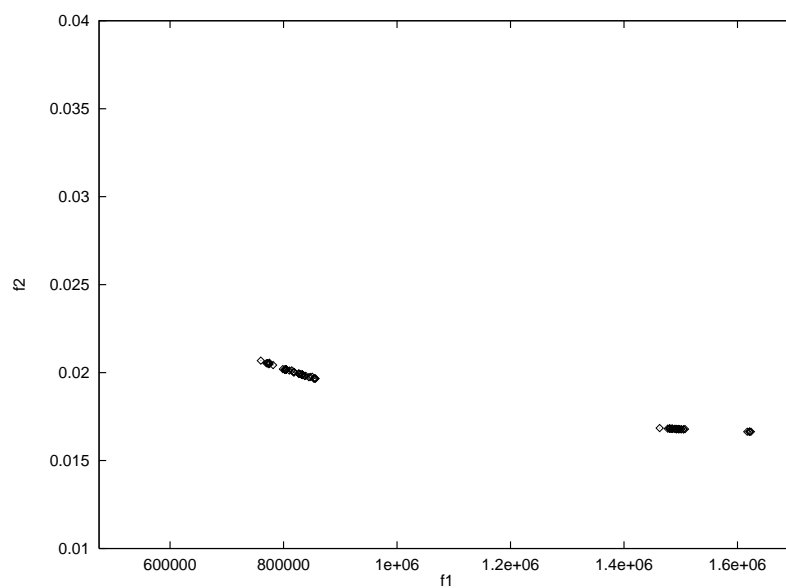


Figure 7.7: Distribution of points in the objective function domain for example 3 using my min-max strategy with a sharing factor of 0.0001 under floating point representation.

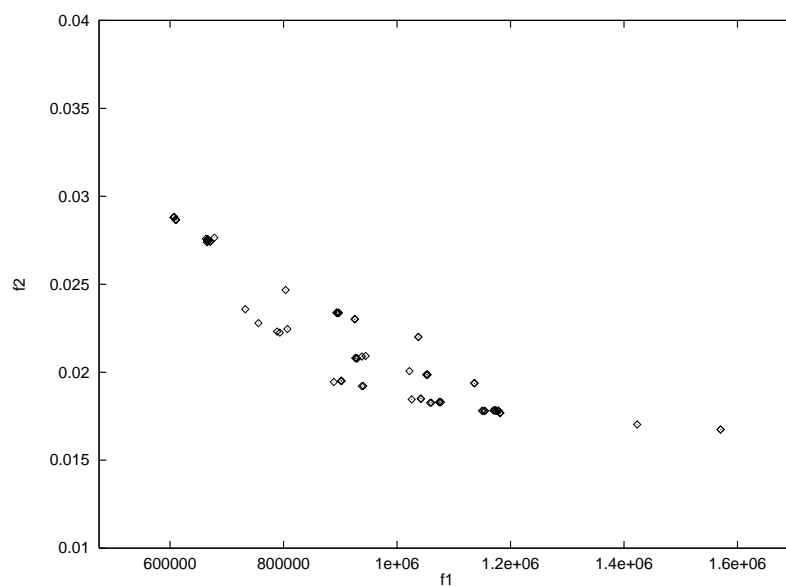


Figure 7.8: Distribution of points in the objective function domain for example 3 using my min-max strategy with a sharing factor of 100.0 under binary representation.

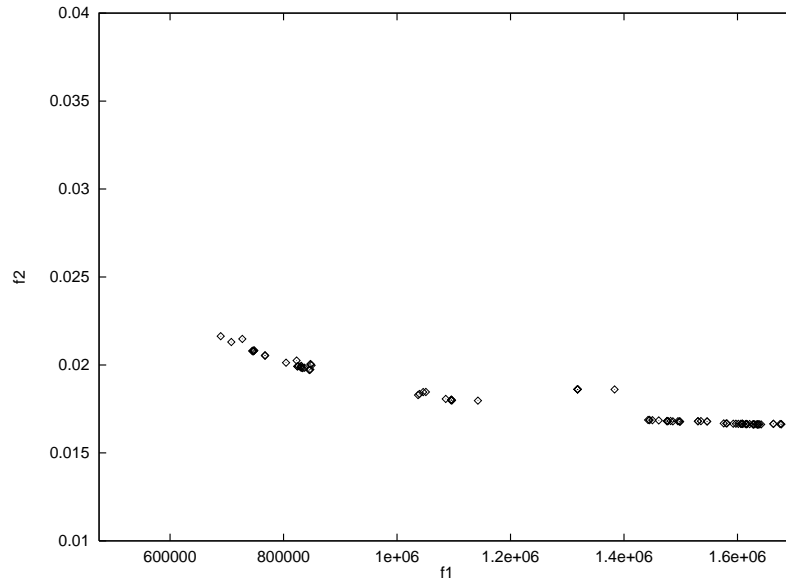


Figure 7.9: Distribution of points in the objective function domain for example 3 using my min-max strategy with a sharing factor of 100.0 under floating point representation.

the first example, floating point representation provides better overall results in general, but poorer distributions than binary representation. A couple of examples should illustrate this. If we look at the graph corresponding to  $\sigma_{share} = 0.0001$  (see Figures 7.2 and 7.3) and  $\sigma_{share} = 100.0$  (see Figures 7.4 and 7.5) we will notice the trend of binary representation to spread the population over the Pareto front, whereas floating point representation normally keeps fewer points and it has more difficulties forming niches. However, it can succeed converging towards the Pareto contour, although it probably will not keep it for too long. This confirms my assertion on the quick convergence property of floating point representation, and explains why this representation scheme produces better overall results than binary representation. For the second example, little analysis is required, since the ideal vector is practically feasible. From the results of Table 7.5, we can see that floating point representation again provides better results in general, but there is a strong trend to converge towards a unique solution. Binary representation, on the other hand, produces better distributions, although it has more difficulties to

Problem	$\sigma_{share}$	$L_p(f)$ (B)	$L_p(f)$ (FP)	$q$
Example 1	22.727	2.653479	2.567664	10
Example 2	208.0	0.291994	0.000012	3
Example 3	46.40	1.660654	1.492613	3
Example 4	1992.38	22.727313	22.489108	10
Example 5	1060.39	704.262255	687.680069	10
Example 6	2486.79	265.368984	215.217609	10
Example 7	14.47	0.859784	3.250367	3

Table 7.7: Comparison of results using estimated values of  $\sigma_{share}$  with my min-max method.  $L_p(f)$  is the maximum deviation with respect to the ideal vector. The value of  $q$  indicated the number of desired peaks to which we want the population to converge.

find the best overall result. In the third example, a variation in the value of  $\sigma_{share}$  does not produce too much difference in the results, unless a large value is used or no sharing is done. Again, floating point representation consistently found better overall results, but with poorer distributions as we can see in Figures 7.6 to 7.9. The conclusion after 3 examples seems clear: we should use binary representation with my method to get the Pareto front, and floating point representation to get the best overall result. The value suggested for  $\sigma_{share}$  when no information about the problem is provided, is 0.5 or 1.0 for binary representation and 0.00001 and 0.5 for floating point representation. That is the reason why I mostly preferred to use 0.5 for both approaches to start experimenting with my method.

After reviewing some literature on niches, besides the seminal work published by Goldberg and Richardson [176], I found extremely interesting the paper by Deb and Goldberg [201], and I decided to use a sharing function over the parameters (instead of the fitness results, as I did in my method). The modification to my code was straightforward, and since I assume that the user always knows the ranges of the variables used, it was very easy to compute the estimate that Deb and Goldberg proposed. The principle to derive such an estimate is to

assume that each niche is enclosed in a  $p$ -dimensional hypersphere of radius  $\sigma_{share}$  such that each sphere encloses  $\frac{1}{q}$  of the volume of the space, where  $q$  is the number of peaks in the solution space. The radius of a hypersphere containing the entire space is calculated as [201]

$$r = \frac{1}{2} \sqrt{\sum_{k=1}^p (x_{k,max} - x_{k,min})^2} \quad (7.2)$$

and the volume is calculated as  $V = cr^p$  with  $c$  a constant. Dividing this volume in  $q$  parts and recognizing that the hypervolume has the same form regardless of size,  $\sigma_{share}$  may be calculated as:

$$\sigma_{share} = \frac{r}{\sqrt[p]{q}} \quad (7.3)$$

I tested this estimate on several of the examples from Chapter 5, and the results are displayed in Table 7.7. With the first 3 examples, the results are very encouraging, since this methodology could find better results with the same population sizes as before. With the other examples, the results were the same, which is indicative of the strong trend of the algorithm to converge towards zones in which one or more elements of the ideal vector have been targeted. This is understandable given the nature of the selection strategy used. The value of  $q$  could vary, and the ones displayed in Table 7.7 are just representative. In most cases, there was not much difference in the results when they were changed. Since we normally do not know how many peaks (or Pareto solutions) the problem has, it is hard to estimate this parameter. However, it is encouraging to see that in some cases, such as in example 2, this approach provided with results that could not be achieved before with larger populations. The only drawback that I see to this approach to establish niches is that the populations generated are normally not very well distributed, even if large values of  $q$  are used. Therefore, my decision was

to consider this approach for getting the best overall result, and use the previous fitness sharing approach to get a better distribution of the population. This makes a lot of sense if we are interested in better distributions in the objective function domain, so that the decision maker can get a broader perspective of the kind of trade-offs that can be made, and also allows to use the GA as a mathematical programming tool whenever necessary, as in cases in which it is possible to find the ideal vector (as in example 2).

Finally, I also read the paper by Samir Mahfoud [210] on sharing and a way to estimate upper bounds for population sizes in terms of certain fixed parameters such as the number of desired classes (or peaks), the probability of crossover, the probability of disruption (a hard number to get), the maximum number of generations and the probability with which we want to ensure that the population is going to keep the given number of desired classes. Mahfoud uses roulette wheel for his analysis, and they recommend such strategy for applications that involve sharing, so one of the extensions to my work will be to incorporate these recommendations. But regardless of that observation, I tried his equation with a couple of examples to see what sort of upper bounds would produce, and using the parameters that I employed in my experiments, Mahfoud's formula suggests population sizes of about 160 for a probability of disruption of 0.5 and 10 classes, but it grows exponentially as the probability of disruption approaches one. For example, if the probability of disruption is increased to 0.8, the population size is increased to about 260. If the probability of disruption goes up to 1, the population size increases to 420. Those estimates are conservative, and somehow I followed them in my experiments, although I was not aware of this work until I concluded my experiments. I just followed a simple empirical estimate on population size after a brief experimentation period.

## 7.3 Incorporating Knowledge About the Domain

The GA was conceived as a “black box” that could search in any environment without need of specific knowledge about it [145]. However, as time passed, researchers realized that in certain domains it was important to hybridize the GA with other techniques to make it compete with other heuristics in terms of performance. For example Sirag and Weisser [211] proposed a model in which the Boltzmann distribution could be incorporated into the GA, so that a simulated annealing procedure could be applied to crossover, inversion and mutation operators, mainly for combinatorial optimization problems.

In a more general way, Schaffer and Morishima [212] proposed the use of some marks along the string to identify loci of good and bad building blocks so that the same principle of survival of the fittest could be applied to the contents of the chromosome string to make the crossover operator depend not only on the string length, but also on its contents. However, their experiments with this new operator (called punctuated crossover) did not conclusively prove its usefulness [213]. Louis continued this work in his dissertation [205] and proposed a masked crossover operator that makes use of certain rules to propagate segments of the string that are producing better offsprings. Also, his results are not totally convincing, because he applies such masks only to a certain kind of design (that involving the production of sequences, such as circuit design) and his DGA (Designer Genetic Algorithm) does not do better than traditional GAs when there is at least partial deception in the problem, which is precisely the kind of problems in which we are interested on having some improvements of the GA's performance.

More within the area of numerical optimization, I have to mention Davis' efforts to encourage the use of hybrid GAs [162] that incorporate knowledge about



the domain so that their performance can be enhanced. He proposed several strategies such as a generate-and-test technique that combined with a real-coded GA produces very satisfying results in numerical optimization applications. Similar work was conducted by Powell and Skolnick [214] [215] [216] with the system called EnGENEous, which is an interdigitation of expert systems and numerical optimization techniques together with a GA, intended for single objective engineering applications.

Also, there is some interesting work on the mixture of connectionist models and GAs to improve evolution through learning [217] using the Hinton and Nowlan model.

My idea of incorporating some knowledge about the domain into the GA initiated with an attempt to use some of the Pareto generation algorithms (namely the Kuhn-Tucker conditions) within some of the GA's operators, particularly crossover. However, the first experiments were not very encouraging, since such conditions were forcing the population through zones extremely restricted without promoting diversity of the individuals, therefore producing premature convergence. Furthermore, the additional information slowed down the performance of the GA in a significant manner. Therefore, I tried to follow an alternative approach. Inspiration came from Fonseca's paper on MOGA [180] which sketches the idea of using a utility function within the GA following the goal attainment method. His method turned out to be a progressive articulation of references in which he assumes the decision maker as an observer that changes the goals across the search process to guide the GA through the correct solutions area. He also mentioned that ideally, such a method would have an expert system to replace the human decision maker, and would be run in a parallel processor, to account for the inefficiencies of the process.

With this idea in mind, I developed my second algorithm for multiobjective optimization. Its basic structure is very simple, since it only replaces the `select()` function so that instead of running a conventional binary tournament, the individuals are chosen based on their closeness to the ideal vector. When there is a tie, then sharing on their fitnesses is performed, in a similar way as in the NPGA. Since the ideal vector has to be known, then the algorithm was modified to be able to compute it in terms of the population available at each run, so that it did not have to be provided by the decision maker. This algorithm works well in problems in which it is not possible to achieve more than one element of the ideal vector while highly conflicting with the others (which unfortunately is a common case in real-world applications). In such cases, nothing can prevent the technique from converging towards trade-offs in which those objectives not so conflicting practically achieve the ideal vector, but those conflicting get really poor values (either too high or too low, depending on what kind of optimization is being performed). This strategy incorporates some knowledge about the domain (in this case, the current ideal vector) to help guide the search, and uses a selection strategy that takes advantage of such knowledge (through a min-max strategy that selects those individuals closest to the ideal vector) to guide the search. Additionally, feasibility constraints are imposed on the crossover and mutation operators, and to the starting population. More work in this respect is necessary, since the results are very good for certain cases, it may be possible to make it work in a more general manner without sacrificing much of its excellent performance and its need of little external information.

## 7.4 An Expert System to Select a Multiobjective Optimization Method

The final part of this chapter describes the development of an expert system called **MOSES Advisor**, which is intended to help the user to choose the most appropriate multiobjective optimization technique according to the characteristics of the problem to be solved and the computer resources available. The knowledge contained in MOSES Advisor was derived from the information provided at the end of Chapter 6. The structure of the system is straightforward, and it consists of a menu containing all the methods contained in MOSES so that the user may choose the one he/she prefers. According to the option selected, some general information about the method will be displayed, explaining its main strengths and weaknesses so that the user may be aware of them. After that, a few questions will be asked concerning specific characteristics of the problem and/or the resources available to solve it, so that some extra advice may be provided in terms of how appropriate that method is to solve the problem in hand. With that information the user should be able to either apply the method in a better way, or change to another method more appropriate to what he/she wants to do. The expert system still has a lot of room for improvement, since several other things additional to the general information provided could be added. For example, some algorithms for computing share factors and population policies could be added to help the designer decide what parameters to use with the GA. MOSES Advisor was written using NASA's expert systems shell called CLIPS.

A sample session with MOSES Advisor follows:

```
Welcome to MOSES Advisor
```

```
An Expert System for Multiobjective Optimization
```

Method to Analyze:

- 1) Monte Carlo Method 1
- 2) Monte Carlo Method 2
- 3) Osyczka's MOS
- 4) GA-Based Linear Combination of Objectives
- 5) Lexicographic Method
- 6) VEGA
- 7) NSGA
- 8) MOGA
- 9) NPGA
- 10) Hajela's Method
- 11) GA with a Weighted Min-max Technique
- 12) GA with Min-max and Sharing

Choose one option: 1

Monte Carlo method 1 could work in your problem if a sufficiently large population is used (> 50 points). This technique is less efficient than method 2 because it has to explore the search space twice. However, it produces better results because of the way in which generates the Pareto set.

Do you want to generate the entire Pareto front? **yes**

Do you know the ideal vector? **yes**

Do you have more than 4 design variables? **no**

Do you have more than 2 objectives? no

Do you have a lot of computer memory available? yes

Are the ranges of the variables too large? no

If you have enough computer resources available, namely computer memory, you should probably also try method 2, since it is faster than method 1, but it requires more memory because it has to store the entire Pareto set.

If you know the ideal vector for your problem, you should try options 10-12 from the main menu to get better results, rather than using Monte Carlo method 1 or 2.

If you want to generate the entire Pareto front, an exploratory technique such as Monte Carlo methods 1 or 2 is not a good choice, because they are able to generate only one point at a time. Look at options 5-12 from the main menu.

This is sample of the way in which MOSES Advisor operates, explaining in general the main characteristics of the method and then advising if one should use it or not based on a few simple questions. This system is intended to be used as a front-end to MOSES, so that the user can decide more easily which method to choose according to the characteristics of the problem. Nevertheless it is always advisable to try at least one mathematical programming technique together with a GA-based technique to get a better approximation to the desired solution.

## Chapter 8

# Conclusions

### 8.1 Contributions

Through these pages I have shown empirical work leading to a better understanding of different strategies for multiobjective optimization. The algorithms incorporated into MOSES are capable of handling design optimization problems with several objectives, according to the needs of the user in terms of quality of the solution and time and architecture available. After an exhaustive set of experiments, I have been able to delineate lines of behavior of several mathematical programming and GA-based methods for multiobjective optimization. An expert system containing the knowledge derived from such experiments was implemented in CLIPS and made available to the users of MOSES. An ad-hoc floating point representation that performs a direct mapping of a real number into a string of integers with the decimal point at a fixed position was proved to be a robust representation scheme for single objective numerical optimization problems, but not very suitable for multiobjective optimization nor single objective optimizations too susceptible to the design scheme, such as the design of circuits.

I provided a very exhaustive bibliographic review of multiobjective optimization methods from both the Operations Research and the GA communities'

perspectives. The most important methods found in this literature survey were implemented in MOSES and they were tested through a set of 8 numerical optimization problems that involve real-world applications. These tests showed the weaknesses and strengths of each one of these methods, and served as a guide to the design maker to decide which method to use under what conditions.

I also proposed two multiobjective optimization methods based on the min-max optimization approach. The first of them provided the best results in all senses, overpassing every other technique consistently. This technique is very robust because it transforms the multiobjective optimization problem into several single optimization problems, and it works very well independently of the representation scheme used, although binary representation provided better Pareto distribution whereas floating point representation provided better overall results. This, by the way, was the general trend in all methods tried, since floating point representation has a demonstrated quick convergence property. This method is very suitable for distributed or parallel systems, since it requires to execute as many processes as objectives we have, but it can also been used in a sequential architecture if the time constraints are not very strong. The structure of the algorithm for this method requires that the user provides a set of weights and the ideal vector. The first requirement can be easily satisfied, and the set of tests presented in this work provide with enough empirical evidence to prove that only a small amount of such weights is necessary to generate excellent results (no more than 20 in all cases). However, if the number of objectives is increased, this set should be increased correspondingly, at least in a linear manner, at a rate of 10 more weights for each new objective added. The second requirement is harder to satisfy, but I also provided a methodology capable of finding the ideal vector of numerical optimization methods with great accuracy. This contradicts the belief

that GAs are not good optimizers, since the results presented in this thesis give empirical evidence of the contrary. The GA consistently surpassed the results produced by any mathematical programming technique used and even the best results previously reported in the literature. Also, it is possible to provide a utility function that contains the desired target values for each objective, in case we do not want to compute the ideal vector. The algorithm will still work well in such cases, although it may not provide results as good as when the ideal vector is provided.

The second technique that I proposed does not require the ideal vector, since it is able to compute it based on the local populations generated. However, to avoid convergence to a single solution, a form of sharing similar to that employed by the NPGA (Niche Pareto Genetic Algorithm) was implemented, but the problem of finding an optimum tournament size was eliminated by using a min-max binary tournament selection strategy. However, the niche sharing factor still has to be provided by the user. In that direction, some work was conducted to try to find a way to automate the computation of this value. After some experimentation, I provided evidence of the importance of a fitness sharing for delineating Pareto contours, and also of the superiority of doing sharing on the parameter values to obtain better overall results with small populations. My second method is very fast and reliable except in cases in which it is possible to find solutions that highly favor more than one objective (namely when some elements of the ideal vector are achievable) but that highly disfavor other objectives. This behavior is common in highly convex search spaces which are unfortunately very common in numerical optimization techniques and more work is required to extend this method to deal with such situations. So far, the only technique that has produced favorable results is using a variation of Eshelman's CHC algorithm



that erases and regenerates part of the population after a certain number of generations or when it got stuck in a local optimum. This approach is valid, but is not the most desirable way of dealing with the difficulties of this technique, so more work in that direction is required.

The two techniques that I developed ensure that only feasible points are produced at generation zero, and the crossover and mutation operators were modified in such a way that this feasibility property is checked at every moment, so that infeasible solutions are never generated by the algorithms. This property makes my approaches unique, since none of the other GA-based techniques analyzed considered such an important issue, mainly because most of the previous work with multiobjective optimization techniques dealt only with unconstrained problems. My techniques were tested also with problems with several constraints (mostly non-linear), providing excellent results (at least for my first min-max strategy). This aspect of the search is important, because at least one of the algorithms tested in this worked converged towards infeasible solutions in one occasion. Finally, the method normally used to handle constraints into the GA was to use a penalty function, such as I have done before in previous published work.

Another point covered in this thesis was the importance of choosing adequate population sizes to run the genetic algorithm. Some analysis in that direction was performed, showing that reasonably small populations can be used for numerical optimization if a floating point representation is adopted. This is because floating point representation requires strings of shorter length than binary representation. The use of a floating point representation together with a sharing strategy on the parameter values in my second min-max technique produces outstanding results if what we are interested on is the best overall solution (that is

normally what researchers look for in many engineering applications), even when small populations are used.

It should be also mentioned that the circuit design problem (Example 8) required some extra effort in all respects, because of its peculiar characteristics. The concept of design is taken in a broader sense in this case, since a certain arrangement of parts has to be devised. In this application a floating point representation appeared to be too unstable to provide good results, but binary representation found results better than those reported in the literature. This problem was solved with MOSES in a single-optimization mode, although a second objective was added to the fitness function whenever feasible solutions were found, so that those minimizing the number of gates used could be rewarded. The design of circuits has traditionally been considered more within the domain of genetic programming, but some of my recent work shows that it can be also done with traditional genetic algorithms if a proper representation scheme is adopted, and my strategy for adjusting the parameters of the GA is employed.

The last aspect of my work was concerned with the incorporation of knowledge about the domain into the GA. In that respect, I reviewed several references that cover that topic, indicating how my second min-max technique uses some knowledge about the domain (namely the best solutions found for each objective at a given stage of the search) to generate the ideal vector, in a way similar to what Fonseca proposed in his work on MOGA (Multiple Objective Genetic Algorithm). I also experimented with some variation of the min-max procedure and a incorporation of the Kuhn-Tucker conditions directly into the crossover operator, but that did not work very well since the search appeared to be extremely constrained to allow the GA to find good Pareto contours. That is the reason why I decided instead to focus my work on the use of the min-max concept into

the selection strategy in one case (method 2), and into the fitness function in the other (method 1).

Finally, my expert system (the so-called **MOSES Advisor**) also has much room for improvement. It is highly desirable to add to it some features that allow the user to get estimates of the most appropriate sharing factors and population policies suggested, according to the information provided in this thesis, together with some tips and hints about how to tackle problems with certain characteristics, based on previous experiences. In that sense, it would be very useful to enhance the knowledge base with the experiences derived from further experiments with MOSES, so that novice users can take advantage of that and make a better use of the system.

My main goal of providing a system that allows experimentation with different multiobjective optimization techniques and that includes at least one method that is robust under all circumstances (in this case my first min-max strategy) has been satisfied. However, more work is still desirable in my second strategy, so that similar results may be achieved with only a single run of the GA. Hopefully, this tool should be of any use for designers who deal with real-world problems.

## 8.2 Future Work

Much additional work remains to be done, since this is a very broad area of research. For example, it is desirable to do more theoretical work on niches and population sizes for multiobjective optimization problems to verify my empirical results. In that sense, I expect that some of my examples may be useful as benchmarks for those interested in this area. To talk about convergence in this context seems a rather difficult task, since there is no common agreement on what

optimum really means. However, if we use concepts from Operations Research such as the min-max optimum, it should be possible to develop such a theory of convergence for these kinds of problems. Also, it is highly desirable to be able to find more ways of incorporating knowledge about the domain into the GA, as long as it can be automatically assimilated by the algorithm during its execution and does not have to be provided by the user (to preserve its generality). It is also important to follow Eshelman and Schaffer's work on a theoretical framework for the excellent performance of real-coded GAs so that practice can finally meet theory in numerical optimization problems.

# Bibliography

- [1] A. Osyczka, "Multicriteria optimization for engineering design," in *Design Optimization* (J. S. Gero, ed.), pp. 193–227, Academic Press, 1985.
- [2] A. Osyczka, *Multicriterion Optimization in Engineering with FORTRAN programs*. Ellis Horwood Limited, 1984.
- [3] V. Pareto, *Cours D'Economie Politique*, vol. I and II. F. Rouge, Lausanne, 1896.
- [4] H. Baier, "Über algorithmen zur emittlung und charakterisierung pareto-optimaler losungen bei entwurfsaufgaben elastischer tragwerke," *ZAMM*, vol. 57, no. 22, pp. 318–320, 1977.
- [5] L. Duckstein, "Multiobjective optimization in structural design: The model choice problem," in *New Directions in Optimum Structural Design* (E. Atrek, R. H. Gallagher, K. M. Ragsdell, and O. C. Zienkiewicz, eds.), pp. 459–481, John Wiley and Sons, 1984.
- [6] F. Szidarovszky and L. Duckstein, "Basic properties of MODM problems," in *Classnotes 82-1*, Tucson, Arizona: Department of Systems and Industrial Engineering, University of Arizona, 1982.
- [7] K. Koski, "Multicriterion optimization in structural design," in *New Directions in Optimum Structural Design* (E. Atrek, R. H. Gallagher, K. M. Ragsdell, and O. C. Zienkiewicz, eds.), pp. 483–503, John Wiley and Sons, 1984.
- [8] C. L. Hwang and A. S. M. Masud, "Multiple objective decision-making methods and applications," in *Lecture Notes in Economics and Mathematical Systems*, vol. 164, New York: Springer-Verlag, 1979.
- [9] H. Jutler, "Liniejnaja model z nieskolkimi celevymi funkcjami (linear model with several objective functions)," *Ekonomika i matematiceckije Metody*, vol. 3, pp. 397–406, 1967. (In Polish).
- [10] R. Solich, "Zadanie programowania liniowego z wieloma funkcjami celu (linear programming problem with several objective functions)," *Przegląd Statystyczny*, vol. 16, pp. 24–30, 1969. (In Polish).

- [11] A. Osyczka, "An approach to multicriterion optimization problems for engineering design," *Computer Methods in Applied Mechanics and Engineering*, vol. 15, pp. 309–333, 1978.
- [12] A. Osyczka, "An approach to multicriterion optimization for structural design," in *Proceedings of International Symposium on Optimal Structural Design*, University of Arizona, 1981.
- [13] S. Rao, "Game theory approach for multiobjective structural optimization," *Computers and Structures*, vol. 25, no. 1, pp. 119–127, 1986.
- [14] C. H. Tseng and T. W. Lu, "Minimax multiobjective optimization in structural design," *International Journal for Numerical Methods in Engineering*, vol. 30, pp. 1213–1228, 1990.
- [15] S. H. Ibáñez, *Métodos de Diseño Optimo de Estructuras*. Colegio de Ingenieros de Caminos, Canales y Puertos, 1990. (in Spanish).
- [16] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability* (J. Neyman, ed.), (Berkeley, California), pp. 481–492, University of California Press, 1951.
- [17] T. C. Koopmans, "Analysis of production as an efficient combination of activities," in *Activity Analysis of Production and Allocation*, *Cowles Commission Monograph No. 13* (T. C. Koopmans, ed.), (New York, New York), pp. 33–97, John Wiley and Sons, 1951.
- [18] H. Terry, "Comparative evaluation of performance using multiple criteria," *Management Science*, vol. 9, no. 3, pp. 431–442, 1963.
- [19] K. C. Kapur, "Mathematical methods of optimization for multiobjective transportation systems," *Socio-economic planning science*, vol. 4, pp. 451–467, 1970.
- [20] B. Roy, "Problems and methods with multiple objective functions," *Mathematical programming*, vol. 1, no. 2, pp. 239–266, 1971.
- [21] D. P. Loucks, "Conflict and choice: Planning for multiple objectives," in *Economy wide Models and Development Planning* (C. Blitzer, P. Clark, and L. Taylor, eds.), (New York, New York), Oxford University Press, 1975.
- [22] J. L. Cohon and D. H. Marks, "A review and evaluation of multiobjective programming techniques," *Water Resources Research*, vol. 11, pp. 208–220, apr 1975.

- [23] A. P. Wierzbicki, "A methodological guide to multiobjective optimization," in *IIASA Working Paper, WP-79-122*, (Laxenburg, Austria), International Institute for Applied System Analysis, 1979.
- [24] C. L. Hwang, S. R. Paidy, and K. Yoon, "Mathematical programming with multiple objectives: A tutorial," *Computing and Operational Research*, vol. 7, pp. 5–31, 1980.
- [25] J. P. Ignizio, *Linear Programming in Single- and Multiple-Objective Systems*. Prentice-Hall, Inc., 1982.
- [26] A. Osyczka and J. Koski, "Selected works related to multicriterion optimization methods for engineering design," in *Proceedings of Euromech Colloquium*, (University of Siegen), 1982.
- [27] W. Stadler, "A Survey of Multicriteria Optimization or the Vector Maximum Problem, Part I : 1776-1960," *Journal of Optimization Theory and Applications*, vol. 29, pp. 1–52, sep 1984.
- [28] M. K. Starr and M. Zeleny, "MCDM-state and future of the arts," in *Multiple Criteria Decision Making* (M. K. Starr and M. Zeleny, eds.), vol. 6 of *TIMS Studies in the Management Sciences*, pp. 5–29, Amsterdam: North-Holland Publishing Company, 1977.
- [29] E. R. Lieberman, "Soviet multi-objective mathematical programming methods: An overview," *Management Science*, vol. 37, pp. 1147–1165, sep 1991.
- [30] G. W. Evans, "An overview of techniques for solving multiobjective mathematical programs," *Management Science*, vol. 30, pp. 1268–1282, nov 1984.
- [31] P. C. Fishburn, "A survey of multiattribute/multicriterion evaluation theories," in *Multiple Criteria Problem Solving* (S. Zionts, ed.), (Berlin), pp. 181–224, Springer-Verlag, 1978.
- [32] R. Benayoun, B. Roy, and B. Sussman, "Electre: Une méthode pour guider le choix en présence de points de vue multiple," *Direction Scientifique*, 1966. Note de Travail, No. 49.
- [33] J. L. Cohon, *Multiobjective Programming and Planning*. Academic Press, 1978.
- [34] Z. Lounis and M. Z. Cohn, "Multiobjective optimization of prestressed concrete structures," *Journal of Structural Engineering*, vol. 119, pp. 794–808, mar 1993.

- [35] D. G. Carmichael, "Computation of pareto optima in structural design," *International Journal for Numerical Methods in Engineering*, vol. 15, pp. 925–952, 1980.
- [36] L. M. Boychuk and V. O. Ovchinnikov, "Principal methods of solution of multicriterial optimization problems (survey)," *Soviet Automatic Control*, vol. 6, pp. 1–4, 1973.
- [37] M. E. Salukvadze, "On the existence of solution in problems of optimization under vector valued criteria," *Journal of Optimization Theory and Applications*, vol. 12, no. 2, pp. 203–217, 1974.
- [38] P. L. Yu, "Decision dynamics with an application to persuasion and negotiation," in *Multiple Criteria Decision Making* (M. K. Starr and M. Zeleny, eds.), pp. 159–177, New York: North-Holland Publish. Co., 1977.
- [39] L. Duckstein and S. Opricovic, "Multiobjective optimization in river basin development," *Water Resources Research*, vol. 16, pp. 14–20, feb 1980.
- [40] J. Koski, "Multicriterion optimization in structural design," in *Proceedings of International Symposium on Optimum Structural Design*, (University of Arizona, Tucson, Arizona), 1981.
- [41] M. Zeleny, "Adaptive displacement of preferences in decision making," in *Multiple Criteria Decision Making* (M. K. Starr and M. Zeleny, eds.), vol. 6 of *TIMS Studies in the Management Sciences*, pp. 147–157, Amsterdam: North-Holland Publishing Company, 1977.
- [42] A. P. Wierzbicki, "On the use of penalty functions in multiobjective optimization," in *Proceedings of the International Symposium on Operations Research*, (Mannheim, Germany), 1978.
- [43] A. P. Wierzbicki, "The use of reference objectives in multiobjective optimization," in *Multiple Criteria Decision Making Theory and Application* (G. Fandel and T. Gal, eds.), pp. 469–486, New York: Springer-Verlag, 1980.
- [44] M. Zeleny, "Compromise programming," in *Multiple Criteria Decision Making* (M. K. Starr and M. Zeleny, eds.), Columbia, South Carolina: University of South Carolina Press, 1973.
- [45] M. Zeleny, *Multiple Criteria Decision Making*. New York: McGraw-Hill Book Company, 1982.
- [46] A. Charnes and W. W. Cooper, *Management Models and Industrial Applications of Linear Programming*, vol. 1. John Wiley, New York, 1961.



- [47] Y. Ijiri, *Management Goals and Accounting for Control*. North-Holland, Amsterdam, 1965.
- [48] Y. Y. Haimes, W. Hall, and H. Freedman, *Multi-Objective Optimization in Water Resources Systems: The Surrogate Trade-Off Method*. Elsevier, Amsterdam, 1975.
- [49] J. P. Ignizio, *Goal Programming and Extensions*. Heath, Lexington, Massachusetts, 1976.
- [50] J. P. Ignizio, "The determination of a subset of efficient solutions via goal programming," *Computing and Operations Research*, vol. 3, pp. 9–16, 1981.
- [51] A. Goicoechea, D. R. Hansen, and L. Duckstein, *Multiobjective Analysis with Engineering and Business Applications*. New York: John Wiley and Sons, 1982.
- [52] S. S. Rao, "Multiobjective optimization in structural design with uncertain parameters and stochastic processes," *AIAA Journal*, vol. 22, pp. 1670–1678, nov 1984.
- [53] A. Charnes, W. W. Cooper, R. J. Niehaus, and A. Stedry, "Static and dynamic assignment models with multiple objectives and some remarks on organization design," *Management Science*, vol. 15, no. 8, pp. B365–B375, 1969.
- [54] S. Lee and V. Jaaskelainen, "Goal programming: Management's math model," *Industrial Engineering*, pp. 30–35, feb 1971.
- [55] S. Lee, *Goal Programming for Decision Analysis*. Auerbach, Philadelphia, 1972.
- [56] S. S. Rao, "Game theory approach for multiobjective structural optimization," *Computers and Structures*, vol. 25, no. 1, pp. 119–127, 1987.
- [57] J. Nash, "The bargaining problem," *Econometrica*, vol. 18, pp. 155–162, 1950.
- [58] J. Nash, "Two-person cooperative games," *Econometrica*, vol. 21, pp. 128–140, 1953.
- [59] F. Szidarovszky, I. Bogardi, and L. Duckstein, "Use of cooperative games in a multi-objective analysis of mining and environment," in *Proceedings of the 2nd International Conference on Applied Numerical Modelling*, (Madrid, Spain), 1978.

- [60] M. Gershon, L. Duckstein, and A. Bardossy, "Differential dynamic programming application to multi-objective decision making," in *Proceedings of the CORS/ORSA/TIMS Joint Meeting*, (Toronto, Canada), 1981.
- [61] K. W. Hipel and N. M. Fraser, "Metagame analysis of the garrison conflict," *Water Resources Research*, vol. 16, pp. 627–637, aug 1980.
- [62] K. W. Hipel, R. K. Ragade, and T. E. Unny, "Metagame analysis of water resources conflicts," *Journal of the Hydraulic Division of the American Society of Civil Engineers*, vol. HY10, no. 100, pp. 1437–1455, 1974.
- [63] N. Howard, *Paradoxes of Rationality, Theory of Metagames and Political Behaviour*. Cambridge, Mass.: MIT Press, 1971.
- [64] K. W. Hipel, R. K. Ragade, and T. E. Unny, "Metagame theory and its applications to water resources," *Water Resources Research*, vol. 12, pp. 331–339, jun 1976.
- [65] N. M. Fraser and K. W. Hipel, "Solving complex conflicts," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-9, pp. 805–816, dec 1979.
- [66] N. M. Fraser and K. W. Hipel, *Conflict Analysis: Models and Resolution*. New York: Elsevier Science Publishing, 1984.
- [67] K. W. Hipel, M. Wang, and N. M. Fraser, "Hypergame analysis of the falkland/malvinas conflict," *International Studies Quarterly*, vol. 32, pp. 335–358, 1988.
- [68] M. A. Takahashi, N. M. Fraser, and K. W. Hipel, "A procedure for analyzing hypergames," *European Journal of Operational Research*, vol. 18, pp. 111–122, 1984.
- [69] N. Okada, K. W. Hipel, and Y. Oka, "Hypergame analysis of the lake biwa conflict," *Water Resources Research*, vol. 21, pp. 917–926, jul 1985.
- [70] M. Wang, K. W. Hipel, and N. M. Fraser, "Modeling misperceptions in games," *Behavioral Science*, vol. 33, pp. 207–223, 1985.
- [71] J. V. Neuman and O. Morgenstern, *Theory of Game and Economic Behavior*. Princeton, New Jersey: Princeton University Press, second ed., 1947.
- [72] R. L. Keeney, "Multi-dimensional utility functions: Theory, assessment and applications," Operations Research Center 43, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1969.

- [73] H. Raiffa, "Preferences for multi-attributed alternatives," 1969.
- [74] J. O. Berger, *Statistical Decision Theory: Foundations, Concepts and Methods*. New York: Springer-Verlag, 1980.
- [75] R. L. Keeney and H. Raiffa, *Decision with Multiple Objectives: Preferences and Value Trade-offs*. New York: John Wiley and Sons, 1976.
- [76] K. R. Oppenheimer, "A proxy approach to multi-attribute decision making," *Management Science*, vol. 24, pp. 675-689, feb 1978.
- [77] T. W. Keelin, *A protocol and procedure for assessing multi-attribute preference functions*. Dept. of engineering economic systems, Stanford University, Stanford, California, 1976.
- [78] 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*, vol. 19, no. 4, pp. 357-368, 1972.
- [79] R. Krzysztofowicz and L. Duckstein, "Preference criterion for flood control under uncertainty," *Water Resources Research*, vol. 15, no. 3, pp. 513-520, 1979.
- [80] Y. Y. Haimes and W. A. Hall, "Multiobjectives in water resources systems analysis: The surrogate trade-off method," *Water Resources Research*, vol. 10, pp. 615-624, aug 1974.
- [81] Y. Y. Haimes, P. Das, and K. Sung, "Multi-objective analysis in the maumee river basin: A case study on level b planning," Tech. Rep. SED-WRG-77-1, Case Western Reserve University, Cleveland, Ohio, 1977.
- [82] Y. Y. Haimes, *Hierarchical Analysis of Water Resource Systems: Modeling and Optimization of Large-Scale Systems*. New York: McGraw-Hill International Book Co., 1977.
- [83] P. Das and Y. Y. Haimes, "Multiobjective optimization in water quality and land management," *Water Resources Research*, vol. 15, pp. 1313-1322, dec 1979.
- [84] S. C. Olenik and Y. Y. Haimes, "A hierarchical multi-object method for water resources planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-9, no. 9, pp. 534-544, 1979.
- [85] L. David and L. Duckstein, "Multi-criterion ranking of alternative long-range water resource systems," *Water Resources Bulletin*, vol. 12, no. 4, pp. 731-745, 1976.

- [86] L. Duckstein and M. Gershon, "Multi-objective analysis of a vegetation management problem using electre ii," Tech. Rep. 81-11, Department of Systems and Industrial Engineering, University of Arizona, Tucson, Arizona, 1981.
- [87] P. Rietveld, *Multiple Objective Decision Methods and Regional Planning*. New York: North-Holland, 1980.
- [88] L. Duckstein, M. Gershon, and R. McAniff, "Development of the santa cruz river basin: A comparison of multi-criterion approaches," Tech. Rep. 51, Engineering Experimentation Station, University of Arizona, Tucson, Arizona, 1981.
- [89] B. Roy, "Electre III: Algorithme de classement basé sur une représentation floue des préférences en présence de critères multiples," *Cahiers du C.E.R.O.*, vol. 20, no. 1, pp. 3–24, 1978.
- [90] J. M. Skalka, "Electre III et IV. aspects méthodologiques et guide d'utilisation," *Document 25*, 1984. Lamsade, Paris.
- [91] B. Roy, "Classement et choix en présence de points de vue multiples (la méthode ELECTRE)," *Revue française d'Informatique et de Recherche Opérationnelle*, vol. 6, no. 8, pp. 57–75, 1968.
- [92] B. Roy and P. Bertier, "La méthode electre: Une application du média planning," in *Operational Research* (M. Ross, ed.), pp. 291–302, Amsterdam: North-Holland, 1973.
- [93] J. Kempf, L. Duckstein, and J. Casti, "Polyhedral dynamics and fuzzy sets as a multi-objective decision making aid," in *Proceedings of the Joint Nat'l TIMS/PRSA Meeting*, (New Orleans, Louisiana), 1979.
- [94] L. Duckstein and J. Kempf, "Multi-criteria q analysis for plan evaluation," in *Proceedings of the 9th Meeting of the EURO Working Group on MCDM*, (Amsterdam), 1979.
- [95] R. Pfaff and L. Duckstein, "Ranking alternative plans that manage the santa cruz river basin by using q-analysis as a multi-criteria decision-making aid," in *Proceedings of the Joint AZ Sect. AWRA, and Hydrology Sect.*, (AZ-Nevada Acad. of Science, Tucson, Arizona), 1981.
- [96] R. G. Atkin, "An approach to structure in architectural and urban design," *Environment and Planning Bulletin*, vol. 1, pp. 51–67, 1974.
- [97] J. P. Brans, P. Vincke, and B. Mareschal, "How to select and how to rank projects: the PROMETHEE method," *European Journal of Operational Research*, vol. 24, pp. 228–238, feb 1986.

- [98] J. P. Brans and P. Vincke, "A preference ranking organisation method (the PROMETHEE method for multiple criteria decision-making)," *Management Science*, vol. 31, pp. 647–656, jun 1985.
- [99] G. D'Avignon, M. Turcotte, L. Beaudry, and Y. Duperre, "Degré de spécialisation des hôpitaux de quebec," tech. rep., Université Laval, Quebec, Canada, jul 1983.
- [100] J. M. Dujardin, "Une évaluation multicritère de projets de remédiation à l'échec dans l'enseignement secondaire belge," in *XIX Meeting of the European Working Group on Multiple Criteria Decision Aid*, (Liège, France), mar 1984.
- [101] B. Mareschal and J.-P. Brans, "Geometrical representations for MCDA," *European Journal of Operational Research*, vol. 34, pp. 69–77, feb 1988.
- [102] G. V. Huylenbroeck, "The conflict analysis method: bridging the gap between ELECTRE, PROMETHEE and ORESTE," *European Journal of Operational Research*, vol. 82, pp. 490–502, may 1995.
- [103] M. Roubens, "Preference relations on actions and criteria in multicriteria decision making," *European Journal of Operational Research*, vol. 10, pp. 51–55, 1982.
- [104] R. Bellman and S. Dreyfus, *Applied Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1962.
- [105] R. E. Larson and J. Casti, *Principles of Dynamic Programming, Part I: Basic Analytic and Computational Methods*. New York: Marcel Dekker, Inc., 1978.
- [106] P. L. Yu and G. Leitmann, "Compromise solutions, domination structures and salukvadze's solution," in *Multi-Criteria Decision Making and Differential Games* (G. Leitmann, ed.), pp. 85–101, New York: Plenum Press, 1976.
- [107] B. Villarreal and M. H. Karwin, "Dynamic programming approaches for multi-criterion integer programming," Tech. Rep. 78-3, Department of Industrial Engineering, SUNY, Buffalo, New York, 1978.
- [108] G. Tauxe, R. Inman, and D. Mades, "Multi-objective dynamic programming: A classic problem redressed," *Water Resources Research*, vol. 15, no. 6, pp. 1398–1402, 1979.
- [109] G. Tauxe, R. Inman, and D. Mades, "Multi-objective dynamic programming with application to a reservoir," *Water Resources Research*, vol. 15, no. 6, pp. 1403–1408, 1979.

- [110] D. E. Bell, "A utility theory approach to preferences for money over time," Operations Research Center 72, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1972.
- [111] D. F. Anderson and J. Rohrbaugh, "Objective function dynamics: Evaluating urban systems through time," *IEEE Transactions*, pp. 458–464, 1979.
- [112] L. G. Mitten, "Preference order dynamic programming," *Management Science*, vol. 21, pp. 43–46, 1974.
- [113] F. Seo and M. Sakawa, "A methodology for environmental systems management: Dynamic application of the nested lagrangian multiplier method," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-9, pp. 794–805, dec 1979.
- [114] S. Opricovic, "An extension of compromise programming to the solution of dynamic multi-criteria problems," in *Proceedings of the 9th IFIP Conference on Optimization Techniques*, (Warsaw, Poland), 1979.
- [115] F. Szidarovszky, "Notes on multi-objective dynamic programming," Tech. Rep. 79-1, Department of Systems and Industrial Engineering, University of Arizona, Tucson, Arizona, 1979.
- [116] F. Szidarovszky and L. Duckstein, "A general framework for dynamic multi-objective programming techniques," Tech. Rep. 81-27, Department of Systems and Industrial Engineering, University of Arizona, Tucson, Arizona, 1981.
- [117] D. Jacobson and D. Mayne, *Differential Dynamic Programming*. New York: Elsevier, 1970.
- [118] D. Murray and S. Yakowitz, "Constrained differential dynamic programming and its application to multi-reservoir control," *Water Resources Research*, vol. 15, no. 5, pp. 1017–1027, 1979.
- [119] A. Bardossy, L. Duckstein, and I. Bogardi, "An efficient solution of multi-objective compromise optimization in water resources by differential dynamic programming," Tech. Rep. 81-25, Department of Systems and Industrial Engineering, University of Arizona, Tucson, Arizona, 1981.
- [120] A. Goicoechea, *A multi-objective stochastic programming model in watershed management*. Dept. of systems and industrial engineering, University of Arizona, Tucson, Arizona, 1977. (Unpublished).
- [121] A. Goicoechea, L. Duckstein, and M. Fogel, "Multiple objectives under uncertainty: An illustrative application of protrade," *Water Resources Research*, vol. 15, pp. 203–210, apr 1979.

- [122] S. Vajda, *Probabilistic Programming*. New York: Academic Press, 1972.
- [123] Y. Haimès, K. Loparo, S. C. Olenik, and S. Nanda, "Multi-objective statistical method for interior drainage systems," *Water Resources Research*, vol. 16, no. 3, pp. 465–475, 1980.
- [124] A. Goicoechea, L. Duckstein, and M. Fogel, "Multi-objective programming in watershed management: A study of the charleston watershed," *Water Resources Research*, vol. 12, pp. 1085–1092, dec 1976.
- [125] R. Benayoun, J. Montgolfier, J. Tergny, and O. Laritchev, "Linear programming with multiple objective functions: Step method (stem)," *Mathematical Programming*, vol. 1, no. 3, pp. 366–375, 1971.
- [126] C. N. Klahr, "Multiple objectives in mathematical programming," *Operations Research*, vol. 6, no. 6, pp. 849–855, 1958.
- [127] D. Savir, "Multi-objective linear programming," Tech. Rep. ORC 66-21, Operations Research Center, University of California, Berkeley, California, 1966.
- [128] C. Maier-Rothe and J. M. F. Stankard, "A linear programming approach to choosing between multi-objective alternatives," in *Proceedings of the 7th Mathematical Programming Symposium*, (The Hague), 1970.
- [129] S. M. Belenson and K. C. Kapur, "An algorithm for solving multicriterion linear programming problems with examples," *Operations Research Quarterly*, vol. 24, no. 1, pp. 65–77, 1973.
- [130] D. E. Monarchi, C. C. Kisiel, and L. Duckstein, "Interactive multiobjective programming in water resources: a case study," *Water Resources Research*, vol. 9, pp. 837–850, aug 1973.
- [131] S. Zionts and J. Wallenius, "An interactive programming method for solving the multiple criteria problem," *Management Science*, vol. 22, no. 6, pp. 652–665, 1976.
- [132] D. E. Monarchi, "Interactive algorithm for multiple objective decision making," Tech. Rep. 6, Hydrology and Water Resources Department, The University of Arizona, Tucson, Arizona, 1972.
- [133] K. R. Oppenheimer, *A Proxy Approach to Multi-Attribute Decision Making*. Dept. of engineering-economic systems, Stanford University, Stanford, California, 1977.
- [134] P. Nijkamp and J. B. Vos, "A multicriteria analysis for water resource and land use development," *Water Resources Research*, vol. 13, pp. 513–518, jun 1977.

- [135] S. S. Rao, "Multi-objective optimization of fuzzy structural systems," *International Journal for Numerical Methods in Engineering*, vol. 24, pp. 1157–1171, 1987.
- [136] A. K. Dhingra, S. S. Rao, and H. Miura, "Multiobjective decision making in a fuzzy environment with applications to helicopter design," *AIAA Journal*, vol. 28, pp. 703–710, apr 1990.
- [137] J.-M. Blin, "Fuzzy sets in multiple criteria decision-making," in *Multiple Criteria Decision Making* (M. K. Starr and M. Zeleny, eds.), vol. 6 of *TIMS Studies in the Management Sciences*, pp. 129–146, Amsterdam: North-Holland Publishing Company, 1977.
- [138] C. H. Coombs, "On the use of inconsistency of preferences in psychological measurement," *Journal of Experimental Psychology*, vol. 5, pp. 1–7, 1958.
- [139] M. Zeleny, "The theory of the displaced ideal," in *Multiple Criteria Decision Making* (M. Zeleny, ed.), pp. 153–206, New York: Springer-Verlag, 1976.
- [140] G. V. Sarma, L. Sellami, and K. D. Houam, "Application of lexicographic goal programming in production planning - two case studies," *Opsearch*, vol. 30, no. 2, pp. 141–162, 1993.
- [141] Y. L. Chen and C. C. Liu, "Multiobjective VAR planning using the goal-attainment method," *IEE Proceedings on Generation, Transmission and Distribution*, vol. 141, pp. 227–232, may 1994.
- [142] C. Darwin, *The Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life*. The Book League of America, 1929. Originally published in 1859.
- [143] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Harbor : University of Michigan Press, 1975.
- [144] J. H. Holland, *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, Massachusetts: MIT Press, 1992.
- [145] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Mass. : Addison-Wesley Publishing Co., 1989.
- [146] J. R. Koza, *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.
- [147] J. Heitkoetter and D. Beasley, "The hitch-hiker's guide to evolutionary computation (faq in comp.ai.genetic)." USENET, sep 1995. (Version 3.3).



- [148] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, second ed., 1992.
- [149] B. P. Buckles and F. E. Petry, *Genetic Algorithms*. Technology Series, IEEE Computer Society Press, 1992.
- [150] D. E. Goldberg and K. Deb, "A comparison of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms* (G. E. Rawlins, ed.), pp. 69–93, San Mateo, California: Morgan Kaufmann, 1991.
- [151] A. K. D. Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.
- [152] L. B. Booker, "Intelligent behavior as an adaptation to the task environment," Tech. Rep. 243, University of Michigan at Ann Arbor, Ann Arbor, Michigan, 1982.
- [153] A. Brindle, *Genetic Algorithms for Function Optimization*. PhD thesis, Department of Computer Science of the University of Alberta, Alberta, Canada, 1981.
- [154] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms* (J. Grefenstette, ed.), (Hillsdale, New Jersey), pp. 14–21, Lawrence Erlbaum Associates, 1987.
- [155] J. J. Grefenstette and J. E. Baker, "How genetic algorithms work: A critical look at implicit parallelism," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, California), pp. 20–27, George Mason University, Morgan Kaufmann Publishers, jun 1989.
- [156] J. E. Baker, "Adaptive selection methods for genetic algorithms," in *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (J. J. Grefenstette, ed.), (Hillsdale, New Jersey), pp. 100–111, Lawrence Erlbaum, 1985.
- [157] D. Whitley, "The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Proceedings of the Third Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, California), pp. 116–121, George Mason University, Morgan Kaufmann Publishers, jun 1989.
- [158] D. E. Goldberg, "Real-coded genetic algorithms, virtual alphabets and blocking," Tech. Rep. 90001, University of Illinois at Urbana-Champaign, Urbana, Illinois, sep 1990.

- [159] A. H. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), pp. 205–218, San Mateo, California: Morgan Kaufmann Publishers, 1991.
- [160] H. P. Schwefel, *Numerical Optimization of Computer Models*. Great Britain: John Wiley and sons, 1981.
- [161] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *Foundations of Genetic Algorithms 2* (L. D. Whitley, ed.), pp. 187–202, San Mateo, California: Morgan Kaufmann Publishers, 1993.
- [162] L. Davis, ed., *Handbook of Genetic Algorithms*. New York, New York: Van Nostrand Reinhold, 1991.
- [163] R. S. Rosenberg, *Simulation of genetic populations with biochemical properties*. PhD thesis, University of Michigan, Ann Harbor, Michigan, 1967.
- [164] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," tech. rep., Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, U. K., 1994.
- [165] W. Jakob, M. Gorges-Schleuter, and C. Blume, "Application of genetic algorithms to task planning and learning," in *Parallel Problem Solving from Nature, 2nd Workshop* (R. Männer and B. Manderick, eds.), Lecture Notes in Computer Science, (Amsterdam), pp. 291–300, North-Holland Publishing Company, 1992.
- [166] B. J. Ritzel, J. W. Eheart, and S. Ranjithan, "Using genetic algorithms to solve a multiple objective groundwater pollution containment problem," *Water Resources Research*, vol. 30, pp. 1589–1603, may 1994.
- [167] P. B. Wilson and M. D. Macleod, "Low implementation cost IIR digital filter design using genetic algorithms," in *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, (Chelmsford, U.K.), pp. 4/1–4/8, 1993.
- [168] H. Adeli and N.-T. Cheng, "Augmented lagrangian genetic algorithm for structural optimization," *Journal of Aerospace Engineering*, vol. 7, pp. 104–118, jan 1994.
- [169] D. Powell and M. M. Skolnick, "Using genetic algorithms in engineering design optimization with non-linear constraints," in *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, ed.), pp. 424–431, Morgan Kaufmann Publishers, jul 1993.

- [170] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pp. 93–100, Lawrence Erlbaum, 1985.
- [171] J. J. Grefenstette, "GENESIS: A system for using genetic search procedures," in *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, pp. 161–165, 1984.
- [172] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard, "Some guidelines for genetic algorithms with penalty functions," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (George Mason University), pp. 191–197, Morgan Kaufmann Publishers, 1989.
- [173] M. P. Fourman, "Compaction of symbolic layout using genetic algorithms," in *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pp. 141–153, Lawrence Erlbaum, 1985.
- [174] F. Kursawe, "A variant of evolution strategies for vector optimization," in *Parallel Problem Solving from Nature. 1st Workshop, PPSN I* (H. P. Schwefel and R. Männer, eds.), vol. 496 of *Lecture Notes in Computer Science*, (Berlin, Germany), pp. 193–197, Springer-Verlag, oct 1991.
- [175] P. Hajela and C. Y. Lin, "Genetic search strategies in multicriterion optimal design," *Structural Optimization*, vol. 4, pp. 99–107, 1992.
- [176] D. E. Goldberg and J. Richardson, "Genetic algorithm with sharing for multimodal function optimization," in *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (J. J. Grefenstette, ed.), pp. 41–49, Lawrence Erlbaum, 1987.
- [177] M. R. Hilliard, G. E. Liepins, M. Palmer, and G. Rangarajen, "The computer as a partner in algorithmic design: Automated discovery of parameters for a multiobjective scheduling heuristic," in *Impacts of Recent Computer Advances on Operations Research* (R. Sharda, B. L. Golden, E. Wasil, O. Balci, and W. Stewart, eds.), New York: North-Holland Publishing Company, 1989.
- [178] G. E. Liepins, M. R. Hilliard, J. Richardson, and M. Palmer, "Genetic algorithms application to set covering and travelling salesman problems," in *Operations research and Artificial Intelligence: The integration of problem-solving strategies* (D. E. Brown and C. C. White, eds.), pp. 29–57, Norwell, Massachusetts: Kluwer Academic, 1990.

- [179] S. M. Mahfoud, "Crowding and preselection revisited," in *Parallel problem Solving from Nature, 2nd Workshop* (R. Männer and B. Manderick, eds.), (Amsterdam), North-Holland Publishing Company, 1992.
- [180] C. M. Fonseca and P. J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," in *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, ed.), (San Mateo, California), pp. 416–423, University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers, 1993.
- [181] N. Srinivas and K. Deb, "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evolutionary Computation*, vol. 2, pp. 221–248, fall 1994.
- [182] N. Srinivas and K. Deb, "Multiobjective optimization using nondominated sorting in genetic algorithms," tech. rep., Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, India, 1993.
- [183] J. Horn and N. Nafpliotis, "Multiobjective Optimization using the Niche Pareto Genetic Algorithm," Tech. Rep. IlliGAL Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, 1993.
- [184] J. G. Lin, "Maximal vectors and multi-objective optimization," *Journal of Optimization Theory and Applications*, vol. 18, pp. 41–64, jan 1976.
- [185] D. M. Himmelblau, *Applied Nonlinear Programming*. New York: McGraw-Hill Book Company, 1972.
- [186] R. Hooke and T. A. Jeeves, "Direct search solution of numerical and statistical problems," *Journal of the ACM*, vol. 8, pp. 221–230, 1961.
- [187] A. Ben-Tal, "Characterization of pareto and lexicographic optimal solutions," in *Multiple Criteria Decision Making Theory and Applications* (G. Fandel and T. Gal, eds.), vol. 177 of *Lecture Notes in Economics and Mathematical Systems*, pp. 1–11, Berlin: Springer-Verlag, 1980.
- [188] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, pp. 1–16, Spring 1995.
- [189] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.

- [190] L. J. Eshelman, "The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms* (G. E. Rawlins, ed.), pp. 265–283, San Mateo, California: Morgan Kaufmann Publishers, 1991.
- [191] M. Ghiassi, R. E. DeVor, M. I. Dessouky, and B. A. Kijowski, "An application of multiple criteria decision making principles for planning machining operations," *IIE Transactions*, vol. 16, pp. 106–114, jun 1984.
- [192] H. Eschenauer, J. Koski, and A. Osyczka, eds., *Multicriteria Design Optimization*. Berlin, Germany: Springer-Verlag, 1990.
- [193] A. D. Belegundu, *A Study of Mathematical Programming Methods for Structural Optimization*. Dept. of civil and environmental engineering, University of Iowa, Iowa, Iowa, 1982.
- [194] S. Rajeev and C. S. Krishnamoorthy, "Discrete optimization of structures using genetic algorithms," *Journal of Structural Engineering*, vol. 118, pp. 1233–50, may 1992.
- [195] B. Armstrong, O. Khatib, and J. Burdick, "The explicit dynamic model and inertial parameters of the PUMA 560 arm," in *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, (San Francisco, California), pp. 510–518, apr 1986.
- [196] J. Koski and A. Osyczka, "Optimal counterweight balancing of robot arms using multicriteria approach," in *Multicriteria Design Optimization. Procedures and Applications* (H. Eschenauer, J. Koski, and A. Osyczka, eds.), ch. 5, pp. 151–167, Berlin, Germany: Springer-Verlag, 1990.
- [197] S. Louis and G. Rawlins, "Designer genetic algorithms: Genetic algorithms in structure design," in *Proceedings of the Fourth International Conference on Genetic Algorithms* (R. K. Belew and L. B. Booker, eds.), (San Mateo, California), pp. 53–60, University of California, San Diego, Morgan Kaufmann Publishers, jul 1991.
- [198] C. A. Coello, M. Rudnick, and A. D. Christiansen, "Using genetic algorithms for optimal design of trusses," in *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, (New Orleans, LA), pp. 88–94, IEEE Computer Society Press, nov 1994.
- [199] C. A. Coello and A. D. Christiansen, "Using genetic algorithms for optimal design of axially loaded non-prismatic columns," in *International Conference on Neural Nets and Genetic Algorithms, ICANNGA '95* (D. W. Pearson, N. C. Steele, and R. F. Albrecht, eds.), (France), pp. 460–463, Ecole des Mines d'Alès, Springer-Verlag, apr 1995.

- [200] C. A. Coello, F. S. Hernández, and F. A. Farrera, "An approach to optimal design of reinforced concrete beams using genetic algorithms," in *Proceedings of the IASTED International Conference on Applied Modelling, Simulation and Optimization* (M. H. Hamza, ed.), (Cancún, México), pp. 141–144, IASTED-ACTA Press, jun 1995.
- [201] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, California), pp. 42–50, George Mason University, Morgan Kaufmann Publishers, jun 1989.
- [202] J. M. Gere and W. Weaver, *Analysis of Framed Structures*. D. Van Nostrand Company, Inc., 1965.
- [203] C. A. Coello, "Análisis de estructuras reticulares por computadora (método de rigideces)." Tesis de Licenciatura, 1991. (in Spanish).
- [204] S. J. Louis, *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Department of Computer Science, Indiana University, aug 1993.
- [205] S. J. Louis and G. J. E. Rawlins, "Pareto optimality, GA-easiness and deception," in *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, ed.), (University of Illinois at Urbana-Champaign), pp. 118–123, Morgan Kaufmann Publishers, 1993.
- [206] D. E. Goldberg, "Optimal initial population size for binary-coded genetic algorithms," Tech. Rep. TCGA Report 85001, University of Alabama, Tuscaloosa, 1985.
- [207] D. E. Goldberg, "Sizing populations for serial and parallel genetic algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, California), pp. 70–79, George Mason University, Morgan Kaufmann, jun 1989.
- [208] C. R. Reeves, "Using genetic algorithms with small populations," in *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, ed.), (San Mateo, California), pp. 92–99, University of Illinois at Urbana Champaign, Morgan Kaufmann, jul 1993.
- [209] J. T. Alander, "On optimal population size of genetic algorithms," in *Proceedings of ComEuro 92*, pp. 65–70, IEEE Computer Society Press, 1992.

- [210] S. W. Mahfoud, "Population size and genetic drift in fitness sharing," in *Foundations of Genetic Algorithms 3* (L. D. Whitley and M. D. Vose, eds.), pp. 185–223, San Francisco, California: Morgan Kaufmann Publishers, 1995.
- [211] D. J. Sirag and P. T. Weisser, "Toward a unified thermodynamic genetic operator," in *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (J. J. Grefenstette, ed.), (Hillsdale, New Jersey), pp. 116–122, Massachusetts Institute of Technology, Lawrence Erlbaum Associates, jul 1987.
- [212] J. D. Schaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms* (J. J. Grefenstette, ed.), (Hillsdale, New Jersey), pp. 36–40, Massachusetts Institute of Technology, Lawrence Erlbaum Associates, jul 1987.
- [213] D. J. Schaffer and A. Morishima, "Adaptive knowledge representation: A content sensitive recombination mechanism for genetic algorithms," *International Journal of Intelligent Systems*, no. 3, pp. 229–246, 1988.
- [214] D. J. Powell, M. M. Skolnick, and S. S. Tong, "Interdigitation : Hybrid technique for engineering design optimization employing genetic algorithms, expert systems, and numerical optimization," in *Handbook of Genetic Algorithms* (L. Davis, ed.), ch. 20, pp. 312–331, New York: Van Nostrand Reinhold, 1991.
- [215] D. Powell, M. Skolnick, and S. Tong, "EnGENEous:domain independent, machine learning for design optimization," in *Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, California), pp. 151–159, George Mason University, Morgan Kaufmann Publishers, jun 1989.
- [216] D. Powell, M. M. Skolnick, and S. Tong, "Using genetic algorithms in engineering design optimization via non-linear constraints," in *Fifth International Conference on Genetic Algorithms*, (University of Illinois at Urbana-Champaign), pp. 424–31, Morgan Kauffman Publishers, jul 1993.
- [217] R. K. Belew, "When both individuals and populations search: Adding simple learning to the genetic algorithm," in *Proceedings of the Third International Conference on Genetic Algorithms* (J. D. Schaffer, ed.), (San Mateo, California), pp. 34–41, George Mason University, Morgan Kaufmann Publishers, jun 1989.

## Biography

Carlos Artemio Coello Coello was born in October 18, 1967 in Tonalá, Chiapas, México. He attended the Escuela de Ingeniería Civil of the Universidad Autónoma de Chiapas, where he obtained a Bachelor of Science in Civil Engineering with Honors in 1991. After that, he decided to go into Computer Science and was admitted to Tulane University's graduate program in the fall of 1991. He obtained a Master of Science degree in Computer Science in 1993, and then decided to continue in the graduate program to get a doctorate. In 1996, he completed his Doctor of Philosophy degree in Computer Science. During all the time that he was at Tulane University, he was supported by a scholarship from the Secretaría de Educación Pública of the Mexican government.