UNIVERSITY OF WARSAW

Faculty of Mathematics, Informatics, and Mechanics

Radhwan Yousif Sedik Al-Jawadi

New Evolutionary Optimization Algorithms Using Similarities and Dissimilarities in Binary Strings

PhD dissertation

Supervisor:

Prof. Dr. hab. Marcin Studniarski

Faculty of Mathematics and Computer Science.

University of Lodz

Warsaw 2018

Author’s declaration:

I hereby declare that this dissertation is my own work.

September 11, 2018 ………………………………………….

Radhwan Al-Jawadi

Supervisor’s declaration:

The dissertation is ready to be reviewed

September 11, 2018 ………………………………………….

Prof. Dr. hab. Marcin Studniarski

Keywords: Genetic algorithm, similarity of chromosomes, dissimilarity of chromosomes, chromosome injection, dynamic schema, free dynamic schema, dynamic dissimilarity, evolutionary algorithms, initial population, schema analysis, convergence of DSC algorithm.

ACM classification 2012

1- Computing methodologies - Machine learning - Machine learning approaches - Bio-inspired approaches - Genetic algorithms.
2- Computing methodologies - Artificial intelligence - Search methodologies - Discrete space search.

Abstract

In this work, six evolutionary algorithms are constructed and programmed by using Graphic User Interface (GUI) in Matlab. They are designed to search for a global optimum of a numerical function. These algorithms are based on exploring similarities and dissimilarities between solutions (chromosomes represented as binary strings) in order to find solutions which are close to an optimal one. Then a special way to discover a schema of a binary string, called free schema, is introduced. The effect of a big initial population is studied in the last algorithm.

To prove the efficiency of these algorithms, twenty seven test functions were used. We used eighteen functions of two variables, one function of four variables, five functions of ten and 100 variables, and five shifted and rotated functions (2, 3-dimentions). The results showed, in most cases, the superiority of the algorithms proposed in this thesis over the Classical Genetic Algorithm (CGA) and some other algorithms like the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and Differential Evolution (DE).

This thesis contains eight chapters. Chapter one is a general introduction to optimization, then a literature review is presented in the second chapter. In the third chapter, an algorithm called Dissimilarity and Similarity of Chromosomes (DSC) is described, which has proved successful in comparison with the CGA. In this algorithm, two genetic operators are used: the dissimilarity operator and the similarity operator, and also random generation of a part of each new population. The DSC succeeded in finding optimum solutions for some functions for which the CGA failed. The fourth chapter introduces a new algorithm that includes two new operators: the dynamic dissimilarity operator and the dynamic schema operator, this algorithm is called DSDSC. The fifth chapter contains descriptions of three new algorithms in which a double population is applied with various genetic processes, including free dynamic schema, these algorithms are named DDS, FDS, and MFDS. In the sixth chapter, the last algorithm, which includes the effect of a big initial population on the MFDS, is constructed, this algorithm is called IPMFDS. Chapter seven contains the comparison of all our methods with CMA-ES, DE and GA; also in addition a case study of the knapsack problem is given here. Finally, chapter eight contains some conclusions of this work.

The proof of convergence is provided only for the DSC algorithm, but it can be easily modified so as to work for all subsequent algorithms. It is suitable for any search that contains random generation of a part of population.

Streszczenie

W niniejszej pracy skonstruowano i zaprogramowano sześć algorytmów ewolucyjnych za pomocą graficznego interfejsu użytkownika (GUI) w programie Matlab. Zostały one zaprojektowane w celu poszukiwania globalnego optimum funkcji numerycznej. Algorytmy te opierają się na badaniu podobieństw i różnic między rozwiązaniami (chromosomy reprezentowane jako łańcuchy binarne) w celu znalezienia rozwiązań bliskich optymalnym. Następnie wprowadzono specjalny sposób wykrywania schematu ciągu binarnego, zwanego wolnym schematem. Wpływ dużej populacji początkowej badany jest w ostatnim algorytmie.

Aby udowodnić skuteczność tych algorytmów, zastosowano 27 funkcji testowych. Zastosowaliśmy 18 funkcji dwóch zmiennych, jedną funkcję czterech zmiennych, 5 funkcji 10 i 100 zmiennych, a także 5 funkcji przesuniętych i obróconych (2 i 3 zmienne). Wyniki pokazały, w większości przypadków, wyższość algorytmów zaproponowanych w tej pracy w stosunku do klasycznego algorytmu genetycznego (CGA) i niektórych innych algorytmów, takich jak strategia adaptacji macierzy kowariancji (CMA-ES) ewolucja różnicowa (DE).

Niniejsza rozprawa zawiera osiem rozdziałów. Rozdział pierwszy jest ogólnym wprowadzeniem do optymalizacji, a następnie przegląd literatury został przedstawiony w drugim rozdziale. W rozdziale trzecim opisano algorytm o nazwie Różnice i Podobieństwa

Chromosomów (DSC), który okazał się skuteczny w porównaniu z CGA. W tym algorytmie używa się dwóch operatorów genetycznych: operatora odmienności i operatora podobieństwa, a także losowego generowania części każdej nowej populacji. DSC odnalazł optymalne rozwiązania dla niektórych funkcji, dla których CGA zawodził. Czwarty rozdział wprowadza nowy algorytm, który zawiera dwa nowe operatory: dynamiczny operator odmienności i operator dynamicznego schematu, który to algorytm nazywa się DSDSC. Piąty rozdział zawiera opisy trzech nowych algorytmów, w których podwójna populacja jest stosowana z różnymi procesami genetycznymi, w tym wolnym schematem dynamicznym, algorytmy te nazywają się DDS, FDS i MFDS. W rozdziale szóstym skonstruowany jest ostatni algorytm, który obejmuje efekt dużej populacji początkowej na MFDS, ten algorytm nazywa się IPMFDS. Rozdział siódmy zawiera porównanie wszystkich naszych metod z CMA-ES, DE i GA; dodatkowo przedstawiono tutaj stadium przypadku dla problemu plecakowego. Wreszcie rozdział siódmy zawiera pewne wnioski z tej pracy.

Dowód zbieżności jest podany tylko dla algorytmu DSC, ale można go łatwo zmodyfikować, aby działał dla wszystkich kolejnych algorytmów. Jest odpowiedni dla każdego wyszukiwania, które zawiera losowe generowanie części populacji.

ACKNOWLEDGEMENTS

First of all, I am grateful to my country, Iraq, especially the Ministry of Higher Education and Scientific Research (MOHESR) and the Engineering Technical College in Mosul for giving me this opportunity to study abroad. Also, I appreciate The Republic of Poland and all staff of the Warsaw University and the University of Lodz, especially my supervisor Prof. Dr. hab. Marcin Studniarski for his valuable insights and wise guidance during the study period, because he showed me valuable guidance in writing the doctoral dissertation and published papers.

I would like to thank my wife Aisha Azeez Younus for her support during the PhD study, as well as also my thanks to my children for their patience with me and their patience of difficulties of alienation.

Eventually, my sincere gratitude goes to my beloved parents (God's mercy on them), my sister and my brothers, for their patience and encouragement throughout the study period. I want to thank all those who helped me out even by a word. I would like to thank all my friends here in Poland and in Iraq.

Radhwan Al-Jawadi

Abbreviations:

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
AES	Average number of Evaluations to a Solution
AI	Artificial Intelligence
AIA	Artificial Immune Algorithms
BA	Bee Algorithm
BBO	Biogeography Based Optimization
BOA	Bayesian Optimization Algorithm
CGA	Classical Genetic Algorithm
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
DDS	Double Dynamic Schema
DE	Differential Evolution
DSC	Dissimilarity and Similarity of Chromosomes
DSDSC	Dynamic Schema with Dissimilarity and Similarity of Chromosomes
EAs	Evolutionary Algorithms
EMO	Evolutionary Multi-objective Optimization
EO	Evolutionary Optimization
ESSE	Extended Stochastic Schemata Exploiter
FDS	Free Dynamic Schema
fGA	forking Genetic Algorithm
FSS	Faure sequence sampling
GA	Genetic Algorithm
GS	Gravitational Search
GUI	Graphical User Interface
HBABC	Hybrid Particle swarm based Artificial Bee Colony
HGA	Homogeneous Genetic Algorithm
HSS	Hamersley sequence sampling

IPMFDS	Initial Population with Multi Free Dynamic Schema
IRPEO	Improved Real-Coded Population-Based Extreme Optimization
LHS	Latin hypercube sampling
MBF	Mean Best Fitness Measure
MFDS	Multi Free Dynamic Schema
MGG	Minimum Generation Gap
NHGA	Non-Homogeneous Genetic Algorithm
NPs	Nested Partitions
NTVPSO	Nonlinear Time-Varying Particle Swarm Optimization
PSO	Particle Swarm Optimization
RBSADE	Ranking Based Self-Adaptive Differential Evolution
RHS	Random Heuristic Search
SC	Soft-Computing
SDVRP	Split Delivery Vehicle Routing Problem
SGA	Simple Genetic Algorithm
SOO	Single Objective Optimization
SSE	Stochastic Schemata Exploiter
TSP	Traveling Salesman Problem

Contents

ACM classification 2012 i

Abstract i

Streszczenie.. ii

ACKNOWLEDGEMENTS iv

Abbreviations: v

Contents vii

List of Tables xii

List of Figures xv

CHAPTER ONE: Introduction 19

1.1 Overview 19

1.2 Theoretical Background of Optimization.. 19

1.3 Principles of Optimization 20

1.3.1 Optimization Techniques (Search Algorithms)............................ 20

1.3.2 Classical Optimization Techniques 22

1.3.3 Advanced Optimization Techniques.. 23

1.3.4 Single Objective Optimization 25

1.4 Genetic Algorithm 26

1.4.1 Genetic Operators... 26

1.5 Aims of the Thesis 27

1.7 Structure of the Thesis 28

CHAPTER TWO: Literature Review 30

2.1 Introduction 30

2.2 What is a metaheuristic? 30

2.3 Some new evolutionary algorithms for SOO 32

2.4 Convergence of genetic algorithms 33

2.5 Repeated runs of a genetic algorithm..... 37

2.6 Genetic algorithms based on schema theory 38

2.7 Poor performance of the GA’s caused by defining length of schemata (messy GA) 39

2.8 Performance measures [53] 40

2.9 Initial population effects 41

2.10 Modified genetic algorithms..... 43

2.11 Hybrid algorithms 45

2.12 Using binary encoding in real-valued function optimization 47

CHAPTER THREE: The DSC Algorithm 51

3.1 Introduction 51

3.2 The idea of similarirty and dissimilarity operators..... 52

3.3 The basics of schema theory 54

3.4 Forma analysis of genetic operators 54

3.4.1 The DSC algorithm..... 58

3.5 Experimental results..... 62

3.6 Discussion of figures..... 76

3.7 Experimental results of GA 80

3.8 Convergence of DSC 82

3.9 Conclusion..... 87

CHAPTER FOUR: The DSDSC Algorithm 89

4.1 Introduction 89

4.2 Methodology of DSDSC algorithm 89

4.3 The DSDSC algorithm 91

4.4 Schema analysis 96

4.5 Experimental results 97

4.6 Conclusion 103

CHAPTER FIVE: The DDS, FDS and MFDS Algorithms 105

5.1 Introduction 105

5.2 Double Dynamic Schema (DDS) algorithm 105

5.2.1 Methodology 106

5.2.1.1 Dissimilarity operator 107

5.2.1.2 Similarity operator 107

5.2.1.3 Dynamic schema operator 107

5.2.1.4 Dynamic dissimilarity operator 108

5.2.2 The DDS algorithm 108

5.2.3 Experimental results 111

5.2.3 Conclusion 120

5.3 Free Dynamic Schema Algorithm (FDS) 120

5.3.1 Experimental results 122

5.4 The Multi Free Dynamic Schema (MFDS) 128

5.4.1 Methodology 130

5.4.2 The MFDS algorithm 132

5.4.3 Experimental results 135

5.4.4 The choice of $\boldsymbol{Ri}$ for Rastrigin, Ackley and Zakharov functions 142

CHAPTER SIX: Initial Population with Multi-Free Dynamic Schema 146

6.1 Introduction 146

6.2 The IPMFDS algorithm 148

6.3 Experimental results ... 151

CHAPTER SEVEN: Comparison of all algorithms on selected continuous and combinatorial optimization problems ... 160

7.1 Comparison of all algorithms ... 160

7.1.1 Comparison of the average number of iterations ... 160

7.1.2 Comparison of the average run time ... 163

7.1.3 Comparison of the number of function evaluations and the success rate ... 167

7.2 Application of all algorithms on some functions from the CEC 2017 benchmark (2- and 3-dimensional shifted and rotated functions) ... 171

7.3 Application of our algorithms to the knapsack problem ... 175

CHAPTER EIGHT: Conclusions ... 178

Appendix A: Test Functions ... 180

A.1 Easom function [122], [123] ... 180

A.2 Matyas function [124] ... 180

A.3 Beale's function [125], [126] ... 180

A.4 Booth's function [127] ... 181

A.5 Goldstein-Price function [122] ... 181

A.6 Schaffer function [128] ... 181

A.7 Schwefel's function [122] ... 181

A.8 Branins's function [122] ... 182

A.9 Six-hump camel back function [122] ... 182

A.10 Shubert's function [122] ... 183

A.11 Martin and Gaddy function ... 183

A.12 Michalewicz function [122] ... 183

A.13 Holder table function [128] ... 183

A.14 Drop wave function [122] 184

A.15 Levy (#13) function [128], [124] 184

A.16 Rastrigin's function 184

A.17 Sum Squares function 185

A.18 Sphere function 185

A.19 Sum of different powers function 185

A.20 Ackley's function [122] 186

A.21 Zakharov function 186

A.22 Rosenbrock's valley function [101] 186

Shifted and rotated test functions (CEC 2017 benchmark) [45].................... 187

A.23 Shifted and Rotated Bent Cigar 187

A.24 Shifted and Rotated Sum of Different Power Function 187

A.25 Shifted and Rotated Zakharov Function 187

A.26 Shifted and Rotated Rosenbrock's Function 188

A.27 Shifted and Rotated Rastrigin's Function 188

References.. 189

List of Tables

Table 3. 1 The dissimilarity operator. 59

Table 3. 2 The similarity operator. 60

Table 3. 3 The results for 50 runs of the DSC algorithm (40 chromosomes). 64

Table 3. 4 The results for 50 runs of the DSC algorithm (80 chromosomes). 67

Table 3. 5 The results for 50 runs of the DSC algorithm (160 chromosomes). 69

Table 3. 6 Comparing the mean number of iterations and success rate of functions for 50 runs of the algorithm (40 vs 80 vs 160 chromosomes). 71

Table 3. 7 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and DSC algorithms (50 runs, max 2500 iterations, 80 chromosomes). 72

Table 3. 8 The number of bits for each function. 74

Table 3. 9 The results for 25 runs of the DSC algorithm for 10-dimensional functions (160 chromosomes) with execution time. 75

Table 3. 10 Comparing the success rate and the mean number of iterations for the GA, first with Bit string and next with Double vector parameter 81

Table 4. 1 All $\boldsymbol{M}$ chromosomes ($\boldsymbol{Ch1..ChM}$). Groups of chromosomes. 90

Table 4. 2 The dynamic schema operator 92

Table 4. 3 The dynamic dissimilarity operator. 93

Table 4. 4 The results for 50 runs of the DSDSC algorithm (80 chromosomes). 98

Table 4. 5 The results for 25 runs of the DSDSC algorithm for 10-dimensional functions with execution time and comparing with GA 101

Table 4. 6 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and DSDSC algorithms (50 runs, max 2500 iterations, 80 chromosomes). 102

Table 5. 1 Populations (P0) and (P1) and the seven groups of chromosomes. 106

Table 5. 2 The results for 50 runs of the DDS algorithm with run time (80 chromosomes). 112

Table 5. 3 The results for 25 runs of the DDS algorithm for 10-dimensional functions with execution time (160 chromosomes)... 114

Table 5. 4 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and DDS algorithms (50 runs, max 2500 iterations, 80 chromosomes)............................. 115

Table 5. 5 The number of function evaluations for 50 runs of the DDS algorithm 116

Table 5. 6 Comparing the average numbers of function evaluations for 50 runs of the DSC, DSDSC and DDS algorithms. 118

Table 5. 7 Comparing the average number of functions evaluations and success rate of BA, PSO and DDS algorithms 119

Table 5. 8 The free dynamic schema operator. 121

Table 5. 9 The results for 50 runs of the FDS algorithm. 123

Table 5. 10 The results for 25 runs of the FDS algorithm for 10-dimensional functions with run time. 125

Table 5. 11 Comparison of CMA-ES, DE and FDS algorithms in terms of mean number of function evaluations and success rate (50 runs, max 2500 iterations, 80 chromosomes). 127

Table 5. 12 Populations (P0) and (P1) and the twelve groups of chromosomes. 131

Table 5. 13 The results for 50 runs of the MFDS algorithm.. 136

Table 5. 14 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and MFDS algorithms (50 runs, max 2500 iterations, 80 chromosomes)........................... 138

Table 5. 15 The results for 25 runs of the MFDS algorithm with execution time of 10-dimensional function. 139

Table 5. 16 The size of $\boldsymbol{Ri}$ parameter on free dynamic schema $(0, \dots, \boldsymbol{mi})$ 143

Table 5. 17 The $\boldsymbol{Ri}$ parameter on free dynamic schema (0 or $\boldsymbol{mi}$). 143

Table 5. 18 Comparing the success rate and mean number of iteration for 25 runs of the MFDS, GA, CMA-ES algorithms on 100-dimensional functions... 144

Table 5. 19 The $\boldsymbol{Ri}$ parameter on free dynamic schema (0 or $\boldsymbol{mi}$). 145

Table 6. 1 The results for 50 runs of the IPMFDS algorithm (80 chromosomes in P0). 152

Table 6. 2 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and IPMFDS algorithms (50 runs, max 2500 iterations, 80 chromosomes) 154

Table 6. 3 The results for 50 runs of the IPMFDS algorithm and run time of 10-dimensional function. 155

Table 6. 4 Comparing the success rate and mean number of iterations on 25 runs of the IPMFDS, GA, CMA-ES algorithms of 100-dimensional functions 156

Table 7. 1 Comparing the average number of iterations for 2-dimensional functions for all algorithms. 161

Table 7. 2 Comparing the average number of iterations for 10-dimensional functions with 160 chromosomes for all algorithms 162

Table 7. 3 Comparing the average run time for 2-dimensional functions for all algorithms. 164

Table 7. 4 Comparing the average of run time for 10-dimensional functions for all algorithms. 166

Table 7. 5 Comparing the average number of function evaluations for 2-dimensional functions with CMA-ES and DE algorithms.. 167

Table 7. 6 The success rate for 2-dimensional functions for all our algorithms comparing with CMA-ES, DE and GA with 80 chromosomes, max. 2500 iterations 169

Table 7. 7 Comparing the number of function evaluation and the success rate for CMA-ES, GA, MFDS and IPMFDS algorithms for 25 runs on 100-dimensional functions. 170

Table 7. 8 The results for 25 runs of all our algorithms comparing with CMA-ES and DE on 2-dimensional of shifted and rotated functions. 172

Table 7. 9 Application of the IPMDSC, CMA-ES and DE algorithms on two 3-dimensional shifted and rotated functions from CEC 2017 with 100 runs. 174

Table 7. 10 The results of knapsack problem with 50 items, for 20 runs of the DSC,DSDSC and GA algorithm (80 chromosomes) by using 150 and 500 iterations. 177

List of Figures

Figure 1. 1 Global and local optimization [1]. 19

Figure 1. 2 Classification of different search techniques. 21

Figure 2. 1 The set of chromosomes available by applying one-point crossover (source: [81], Figure 4.17) 48

Figure 2. 2 The set of chromosomes available by applying the similarity and the dissimilarity operators... 49

Figure 3. 1 A simulation of similarity and dissimilarity idea. 53

Figure 3. 2 Flowchart of the DSC algorithm. 61

Figure 3. 3 The average number of iterations and standard deviation of iterations for 2-dimensional functions with 40 chromosomes for DSC algorithm 66

Figure 3. 4 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for DSC algorithm 68

Figure 3. 5 The average number and standard deviation of iterations for 2-dimensional functions with 160 chromosomes for DSC algorithm 70

Figure 3. 6 The average number and standard deviation of iterations for 10-dimensional functions with 160 chromosomes for DSC algorithm 76

Figure 3. 7 Solutions of Easom Pr... 77

Figure 3. 8 Solutions of Schaffer's problem... 77

Figure 3. 9 Solutions of Holder-table. 77

Figure 3. 10 Solutions of Drop-wave problem. 77

Figure 3. 11 Solutions of Michalewicz pr.. 77

Figure 3. 12 Solutions of Branins's problem... 77

Figure 3. 13: Solutions of Shubert problem: 18 optimal solutions. 78

Figure 3. 14 Solutions of Six-hump camel back problem. 78

Figure 3. 15 Solutions of Levy N.13 pr. 78

Figure 3. 16 Solutions of Schwefel’s Pr. 78

Figure 3. 17 Finding the best solution for Michalewicz problem in 300 iterations. 78

Figure 3. 18 Finding the best solution for Schaffer problem in 140 iterations. 78

Figure 3. 19 shows the Graphical User Interface (GUI) for the DSC algorithm for the Michalewicz function. 79

Figure 4. 1 Flowchart of the DSDSC algorithm. 95

Figure 4. 2 A part of Michalewicz function. 96

Figure 4. 3 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for DSDSC algorithm 99

Figure 4. 4 The GUI of DSDSC algorithm 100

Figure 4. 5 Shubert function with 18 optimum solutions 100

Figure 4. 6 The average number and standard deviation of iterations for 10-dimensional functions with 80 chromosomes for DSDSC algorithm 102

Figure 5. 1 Flowchart of the DDS algorithm. 110

Figure 5. 2 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for DDS algorithm 113

Figure 5. 3 The average number and standard deviation of iterations for 10-dimensional functions with 160 chromosomes for DDS algorithm 115

Figure 5. 4 Comparing the average numbers of function evaluations for DSC, DSDSC and DDS algorithms. 119

Figure 5. 5 Free dynamic schema operator. 122

Figure 5. 6 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for FDS algorithm... 125

Figure 5. 7 The average number and standard deviation of iterations for 10-dimensional functions with 160 chromosomes for FDS algorithm 126

Figure 5. 8 shows the solution of Schaffer N.2 function found in 2 iterations.......................... 128

Figure 5. 9 Multi free dynamic schema. 130

Figure 5. 10 Flowchart of the MFDS algorithm. 134

Figure 5. 11 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for MFDS algorithm.. 137

Figure 5. 12 The average number and standard deviation of iterations for 4- and 10-dimensional functions with 80 chromosomes for MFDS algorithm.. 140

Figure 5. 13 shows the multi free dynamic schema on Michalewicz function finding best solution after 15 iterations (green points belong to the schema). 141

Figure 5. 14 shows the multi free dynamic schema on Michalewicz function finding the best solution (top view, where red points belong to the schema)... 141

Figure 5. 15 shows Schwefel's function, MFDS finding solution in 4 iterations 142

Figure 6. 1 The flowchart of the IPMDS algorithm. 150

Figure 6. 2 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for IPMFDS algorithm.. 153

Figure 6. 3 The average number and standard deviation of iterations for 10-dimensional functions with 80 chromosomes for IPMFDS algorithm 156

Figure 6. 4 Shubert function: one iteration with small range [-4,4]... 158

Figure 6. 5 The behavior of population (P0, P1) after 19 iterations for Michalewicz function. 158

Figure 6. 6 The IPMFDS algorithm has the ability to search in the best area of Michalewicz function. 159

Figure 7. 1 Comparing the average number of iterations for 2-dimensional functions for all algorithms 162

Figure 7. 2 Comparing the average number of iterations for 10-dimensional functions for all algorithms. 163

Figure 7. 3 Comparing the average run time for 2-dimensional functions for all algorithms. ... 165

Figure 7. 4 Comparing the average run time for 10-dimensions functions for all algorithms. .. 166

Figure 7. 5 Comparing the average number of function evaluations for 2-dimensional functions for all algorithms. 168

CHAPTER ONE: Introduction

1.1 Overview

In this chapter we provide an overview of the main optimization methods and principles. We mention some classical methods as well as some new metaheuristic methods, and focus on Genetic Algorithms (GA) and other evolutionary optimization algorithms.

1.2 Theoretical Background of Optimization

The key idea of optimization can be understand from Figure 1. 1, where a function is shown which has various peak values and each one of them can be considered as a type of optimum. There are two main types of peak values: global optima and local optima. The main target in any optimization algorithm is to find a solution (a point x in the domain of f) at which a global optimum is attained.

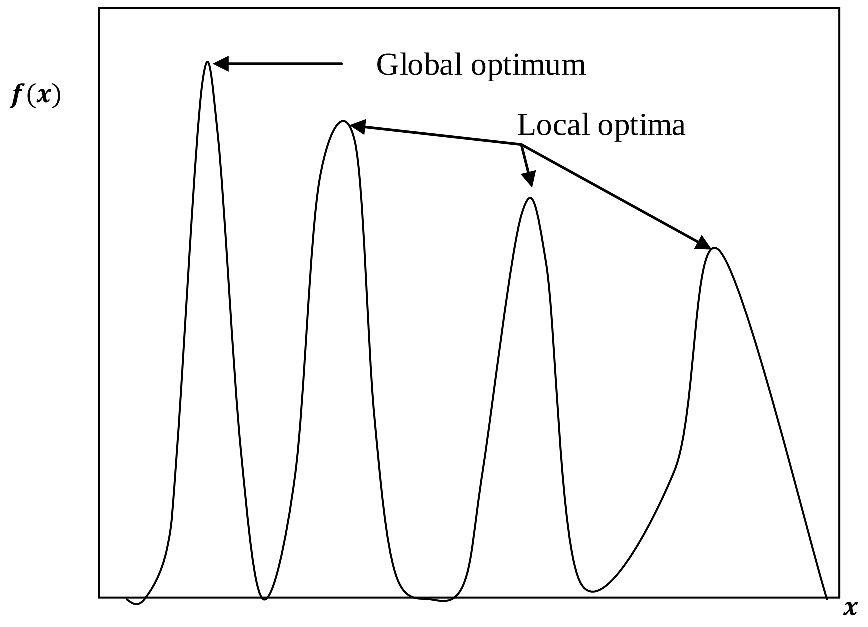


Figure 1. 1 Global and local optimization [1].

1.3 Principles of Optimization

In any problem involving decision making, be it in engineering or in economics, optimization plays a crucial role. The task of decision making entails choosing between various alternatives. Our desire to make the "best" decision stands behind the choice. The goodness of the alternatives is measured by an objective function or performance index.

Optimization theory and techniques deal with selecting the best alternative in the sense of a given objective function. The area of optimization has received enormous attention in recent years, primarily because of the rapid progress in computer technology, including the development and availability of user-friendly software, high-speed and parallel processors, and artificial neural networks [2].

1.3.1 Optimization Techniques (Search Algorithms)

Optimization techniques or, in other words, search algorithms, are one of possible ways to help a decision maker to choose a good solution. Optimization algorithms can also lead to an appropriate solution for real-time applications [3].

In many real world problems, the objectives that are being taken under consideration while trying to find the solution are in conflict with each other, and optimizing a particular solution with respect to only a single objective can result in unacceptable results with respect to other objectives. A reasonable approach to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without allowing one particular objective to dominate [3].

Figure 1.2 illustrates popular search algorithms such as Uninformed Search, Guided Random Search Techniques (Heuristic Search), Calculus Based Techniques, etc. The review [3] provides the comparison and analysis of these algorithms for different problems.

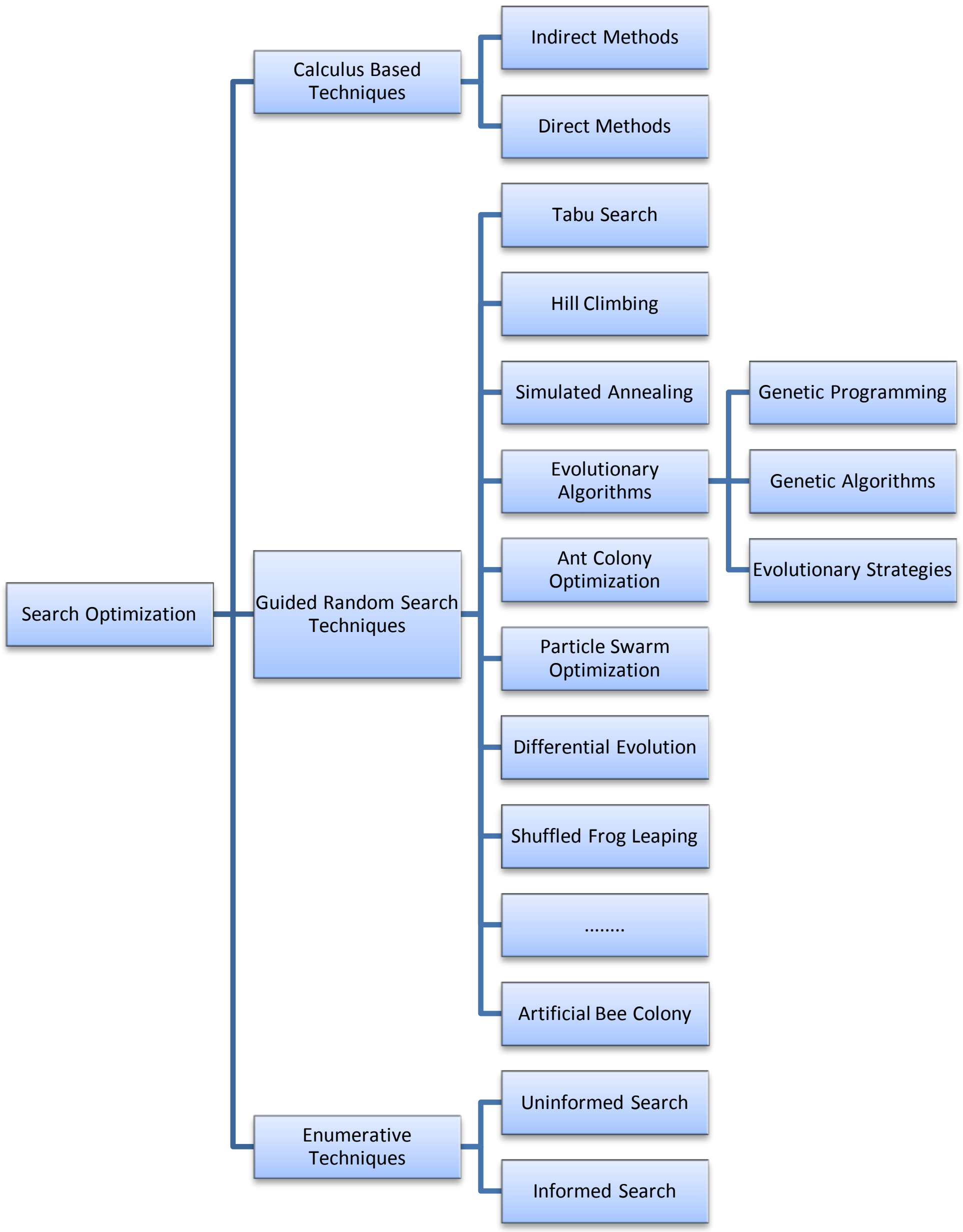


Figure 1. 2 Classification of different search techniques.

Here are some definitions of guided random search algorithms:

1. Hill Climbing is a graph searching algorithm, this algorithm starts with an arbitrary solution and then attempts to find a better one by changing one element of the solution (one bit in the binary encoding). Hill Climbing is

used widely in artificial intelligence fields, for reaching a goal state from a starting node.

2. Simulated Annealing is a probabilistic single-solution-based technique for approximating the global optimum of a given function. The name refers to the technology which includes controlled cooling and heating a material in order to decrease the defects and raise the volumes of its ingredient crystals [3].
3. Genetic Algorithms: A Genetic Algorithm (GA) is a search technique used in computer science to find approximate solutions to optimization and search problems. Specifically, it is generally an incomplete search. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination) [4].
4. Tabu Search was introduced in [5] and [6] to solve combinatorial optimization problems, it has been used effectively for simulation optimization. It is a solution-to-solution method and the main idea is to make certain moves or solutions Tabu, that is they cannot be visited as long as they are on what is called the tabu list. The tabu list L_k is dynamic and after each move, the latest solution θ_k, or the move that resulted in this solution, is added to the list and the oldest solution or move is removed from the list [7].

1.3.2 Classical Optimization Techniques

The classical methods of optimization are usually based on updating a single randomly chosen solution in every iteration by a deterministic procedure, to finally find the optimal one. Those methods can be classified in two distinct groups: direct methods and indirect (gradient-based) methods, see Figure 1.2. To reach an optimal solution, just the constraint functions and the objective function values are utilized in direct techniques. In the case of the indirect methods both values of functions and their gradients are used in the process [8], [9].

The classical methods of optimization are useful in finding the optimum solution of differentiable functions. These methods are analytical and to specify the optimum points the differential calculus strategies can be utilized. Since some of the practical problems involve objective functions that are non-differentiable or even discontinuous, the classical optimization techniques have limited use in practical applications. Yet, the study of these classical techniques of optimization is crucial in the process of developing most of the numerical techniques, which have evolved into advanced techniques more suitable to today's practical problems [10].

There are three main classes of problems that can be handled with the classical optimization techniques, viz., single variable functions, multivariable functions with no constraints and multivariable functions with both equality and inequality constraints. For problems with equality constraints, the Lagrange multiplier method can be used. If the problem has inequality constraints, the conditions of Kuhn-Tucker may be utilized to recognize the optimum solution. These methods lead to a set of nonlinear simultaneous equations that may be difficult to solve [10].

1.3.3 Advanced Optimization Techniques

Since 1960's, more and more attention has been paid to evolutionary methods of optimization which aspire to mimic the fundamentals of nature evolution. This idea has influenced the design of optimization algorithms and stochastic searches [8], [9].

Instead of utilizing a single solution (like in the case of classical methods), evolutionary methods are utilizing sets of random solutions as base populations. To reach the optimal solutions, these base populations are updated in each iteration. Furthermore, evolutionary methods can provide multiple solutions to multi-objective problems [8], [9].

One of the most popular fields of evolutionary computation is the Evolutionary Multi-objective Optimization (EMO), which has proven itself to be successful in various application fields where multiple objectives appear [11].

”Evolutionary Algorithms (EAs) play an important role in the framework of artificial intelligence (AI) and, in particular in Soft-Computing (SC), when dealing with multi-objective problems in real-world engineering optimization” [12].

The Multi-Objective Optimization (MOO), which can also be named Pareto optimization, multi-attribute optimization, multi-criteria optimization, multi-objective programming or vector optimization is a decision making for multiple criteria [13].

Most of the traditional techniques require gradient information and hence it is not possible to solve non-differentiable optimization problems with the help of such traditional techniques. Moreover, such techniques often fail to solve optimization problems that have many local optima. To overcome these difficulties, there is a need to develop more powerful optimization techniques and for the last three decades there has been much effort put to develop these techniques. Some of the well-known population-based optimization techniques are: Genetic Algorithms (GA) [14] which are based on the principle of evolution of the living beings and Darwinian theories of the survival-of-the-fittest; Artificial Immune Algorithms (AIA) [15], based on the principle of immune system of the human being; Ant Colony Optimization (ACO) [16], which mimics the foraging behavior of the ant; Particle Swarm Optimization (PSO) [17] which uses the foraging behavior of the swarm of birds; Differential Evolution (DE) [18] which is similar to GA with specialized crossover and selection method; Shuffled Frog Leaping (SFL) [19] which works on the principle of communication among the frogs, Artificial Bee Colony (ABC) [20] mimicking the principle of foraging behavior of a honey bee. These algorithms have been applied to many engineering optimization problems and proved especially effective in solving some particular problems. All the above-mentioned algorithms are nature inspired population-based optimization methods, but they have some limitations in some aspects [21]. Due to this, more research is required to test algorithms in different situations to check how suitable they are for a wide variety of problems. Research is conducted in order to enhance the existing algorithms and to improve their performance. Enhancement is done either (a) by modifying the existing algorithms or (b) by hybridizing the existing algorithms. Enhancement due to modifications in the existing algorithms is reported in GA [22], [23], PSO [24], [25],

[26], ACO [27], [28], ABC [29], [30], [31], etc. Enhancement can be also done by combining the strengths of different optimization algorithms, such process is known as hybridization of algorithms. Hybridization is an effective way to make the algorithm efficient and it combines the properties of different algorithms. Some of such hybridized algorithms can be found in [32], [33], [34], [35], [36], [37]. For example, a hybrid optimization algorithm combining the Biogeography Based Optimization (BBO) and Artificial Bee Colony (ABC), named Hybrid Particle swarm based Artificial Bee Colony (HBABC), is described in [21]. In [38] a hybrid algorithm is enhanced, it combines with the Nested Partitions (NP) technique.

The NP method is introduced in [39] and it is another metaheuristic for combinatorial optimization that is readily adapted to simulation optimization problems. The main idea of this method lies in systematically partitioning the feasible region into subregions, then evaluating the potential of each region, and eventually focusing the computational effort on the most promising region. This process is carried out iteratively with each partition nested within the last. The computational effectiveness of the NP method relies heavily on the partitioning, which, if carried out in a manner such that fitting solutions are clustered together, can reach a near optimal solution very quickly [7].

In our work, we also use partition of the feasible region into some subregions. It is attained by fixing a number of highest bits in a bit string for each variable. In this way a subregion is defined which is further searched by using genetic operators (see the DSDSC, Chapter 4, and the subsequent algorithms).

1.3.4 Single Objective Optimization

The objective function of a single-objective optimization (SOO) problem may have more than one global optimum point. For instance, the Shubert problem has 18 optimum solutions, see Figure 3. 13, but it is satisfactory if the algorithm reaches any of the solutions. This type of optimization is called singe-objective global optimization.

For example in the Travelling Salesman Problem (TSP), given a list of cities and distance between pairs of cities, the aim is to find the shortest possible route such that each city is visited once and we return to the origin city. In this problem the objective is to minimize the length of the tour [9], [40].

1.4 Genetic Algorithm

Genetic Algorithms (GA) are heuristic search techniques based on the process of natural evolution. They have found applications in generating useful solutions for problems involving optimization and search. Natural selection modeling is the base of GA which does not need computation of any secondary functions like derivatives. Some advantages of GA which make it more useful in optimization problems are the following: (a) the probability of local minimum trapping is decreased (b) going from one state to another requires less computational effort and (c) evaluation of the fitness of each string guides the search. A benefit of using the GA techniques is that they lead, in most of the cases, to global optimal solutions [3].

In 1960s, "Evolutionary computing" was introduced by I. Rechenberg in his work “Evolution strategies”, and was further developed by other researchers. Genetic Algorithms (GAs) were discovered by John Holland who suggested this idea in his book “Adaptation in natural and artificial systems” in 1975 [41]. Holland suggested GA as a heuristic method based on “survival of the fittest”. GA proved to be a useful tool for search and optimization problems [42].

The use of Genetic Algorithms for problem solving is not new. The pioneering work of J. H. Holland in the 1970’s provided a significant contribution for scientific and engineering applications [43].

1.4.1 Genetic Operators

There are usually three operators in a typical GA [44]. The first is the selection operator which produces one or more copies of some individuals from the current population. Individuals with a good fitness are more likely to be chosen; otherwise, the individual is eliminated from the solution pool. Then the second operator is the recombination (known as the “crossover”) operator. In the crossover operator two individuals from the generation and a crossover point are selected and a swapping operation is performed on the bit strings to the right-hand side of the crossover points of both individuals. The crossover operator works for two complementary purposes. First, it provides new points for further testing within the hyperplanes already represented in the population. Furthermore, crossover introduces representatives of new hyperplanes into the population, which has not been represented by either parent structure. Thus the probability of obtaining a better performing offspring is greatly increased. The third operator is the “mutation” operator. This operator acts as a background operator and is used to explore some of the unvisited points in the search space by randomly flipping a “bit” in a population of strings. Frequent application of this operator would lead to a completely random search and because of that is has usually assigned a very low probability of its activation [44].

A genetic search starts with a randomly generated initial population within which each individual is evaluated by means of a fitness function. By using selection, individuals in this and subsequent generations are duplicated or eliminated according to their fitness values. Further generations are created by applying GA operators. This process is designed so as to lead to a generation of highly performing individuals [44].

1.5 Aims of the Thesis

The main purpose of the thesis is to build new evolutionary algorithms which are able to find best solutions of Single Objective Optimization (SOO) problems. These algorithms are tested and compared with the Classical Genetic Algorithm (CGA), Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and Differential Evolution (DE). In our six new algorithms we explore the effect of similarity and dissimilarity of chromosomes in the population, and also effects of discovering the schema. The algorithms are easy to formulate and understand, they were tested on various problems with different types of difficulty. For the first algorithm (DSC), we also study it convergence.

In this thesis, practical experiments were applied on six new evolutionary algorithms (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS), each algorithm applied on two-dimensional and ten-dimensional functions, the last two algorithms applied in 100 dimensions on different functions, also applied on some 3-dimensional shifted and rotated functions taken from CEC 2017 [45]. The results of comparison with other algorithms such as CMA-ES, DE, show in most cases that the new algorithms are superior to find the optimum solution. The number of function evaluations was also calculated.

A beneficial feature of these algorithms is that they do not contain many parameters, only one parameter in the DSC algorithm, which is the number of chromosomes in a population. In the DSDSC, DDS, FDS, MFDS algorithms, we have three parameters: the number of chromosomes in a population, and the minimum and maximum of the values R_i used in the dynamic dissimilarity, dynamic schema and free dynamic schema operators. Then in IPMFDS we have the size of initial population which is also a parameter for the algorithm.

1.7 Structure of the Thesis

The thesis is organized as follows:

Chapter 2: The literature review of single objective optimization, convergence of genetic algorithms, schema theory and initial population effects.

Chapter 3: Presentation of the DSC algorithm, forma analysis and convergence of the DSC algorithm.

Chapter 4: Presentation of the DSDSC algorithm, schema analysis.

Chapter 5: Three new algorithms derived from DSDSC, called DDS, FDS and MFDS.

Chapter 6: A new algorithm is presented which takes advantage of the effect of a big first population, it is applied with multi free dynamic schema and called IPMFDS.

Chapter 7: We apply all algorithms to some shifted and rotated functions taken from CEC 2017 [45], compare all methods with CMA-ES, DE, also in addition a case study of the knapsack problem is presented.

Chapter 8: Conclusions.

Appendix A: Presents all test functions.

CHAPTER TWO: Literature Review

2.1 Introduction

Different approaches based on the Evolutionary Algorithms (EA) technics for solving Single Objective Optimization (SOO) problems, have been proposed in recent years.

In this literature review, we focus on the following topics : metahuristics, single-objective optimization by using Genetic Algorithms (GAs), repeated runs of a GA, the role of a schema in GA, poor performance of GAs, performance measures, the effect of initial population, modified GAs, hybrid algorithms, binary encoding in real-valued function, and convergence of GAs.

Optimization is essential for finding suitable answers to real life problems. In particular, genetic (or more generally, evolutionary) algorithms can provide satisfactory approximate solutions to many problems to which exact analytical results are not accessible.

2.2 What is a metaheuristic?

Global optimization algorithms can be divided into two groups: deterministic algorithms and metaheuristic algorithms, see [46]. Metaheuristic methods are helpful for a wide class of optimization problems where deterministic algorithms are not suitable (for example, functions with a large number of local extrema).

Metaheuristic was firstly mentioned by Fred Glover in 1986 [47]. According to [48], a metaheuristic algorithm is defined as: "An iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions".

Some of the most popular metaheuristic approaches are genetic algorithm, simulated annealing, tabu search, memetic algorithm, ant colony optimization, particle swarm optimization, etc. [49].

Since many real-world optimization problems become increasingly complex, better optimization algorithms are constantly required. Recently, metaheuristic global optimization algorithms become a popular choice for solving complex and loosely defined problems, which would be difficult to solve by traditional methods. Gradient and direct search methods are generally regarded as local search methods. Metaheuristics do not necessarily require a good initial guess, in contrast to both gradient and direct search methods, where an initial guess is highly important for obtaining convergence towards the optimal solution [50].

Metaheuristics require a large number of function evaluations. They are often characterized as population-based stochastic search routines which assure a high probability of escaping the local optimal solutions when compared to gradient-based and direct search algorithms [50].

There are differences between single solution based metaheuristics and population based metaheuristics. The methods of single solution based meta-heuristics include Simulated Annealing, Microcanonic Annealing, Threshold Accepting Method, Noising Method, Tabu Search, Variable Neighborhood Search, Guided Local Search, Iterated Local Search. The methods of population based metaheuristics are as follows [51], [52]:

1. Evolutionary computation: Genetic algorithm, Evolution Strategy, Evolutionary programming, Genetic programming.
2. Swarm intelligence: Ant colony optimization, Particle swarm optimization, Bacterial foraging optimization algorithm, Bee colony optimization-based algorithms, Artificial immune systems, Biogeography-based optimization.
3. Other evolutionary algorithms: estimation of distribution algorithms, differential evolution, Coevolutionary algorithms, cultural algorithms, Scatter Search, Path Relinking.

Evolutionary Algorithms (EAs) constitute a large class of optimization procedures, including classical GAs, that are inspired by the process of natural evolution.

As Eiben and Smith [53] observe, “there are many different variants of evolutionary algorithms. The common underlying idea behind all these techniques is the same: given a population of individuals within some environment that has limited resources, competition for those resources causes natural selection (survival of the fittest)”. Different implementations of EAs (e.g., genetic algorithm, genetic programming, evolutionary strategy) can essentially be summarized by the following steps:

1. Initialize a population randomly and evaluate each candidate;
2. Select parents;
3. Recombine pairs of parents;
4. Mutate the resulting offspring;
5. Evaluate each new candidate;
6. Select individuals for the next generation;
7. Repeat from Step 2 until a stopping criterion is satisfied.

Our algorithms presented in this thesis can be considered as evolutionary algorithms, because they work on the same principles. However, there are two differences. The first one is that our algorithms do not use mutation, but we have included random generation of a part of generation at each iteration; this process, instead of mutation, enhances the diversity of a new population. The second difference is that, instead of selection, we use copying of the best chromosome several times and inserting it in different places of a population.

Moreover, as the authors of [53] notice, “during selection the best individuals are not chosen deterministically, and typically even the weak individuals have some chance of becoming a parent or of surviving”. In our algorithms some of the weak chromosomes become “parents” for genetic operators (like similarity or dissimilarity operator) but the weakest of them are replaced by randomly generated new chromosomes.

2.3 Some new evolutionary algorithms for SOO

Chang et al. [54] propose two new operators which are added to the classical GA: duplication and fabrication. Duplication is a procedure producing multiple copies of the best-fit chromosome from some elite base. It is similar to what has been done in the DSC algorithm (see Chapter 3). The difference is that in [54] the duplicated chromosomes replace the worst chromosomes in the population, while in the DSC algorithm the copies of the best chromosome replace randomly chosen chromosomes. Fabrication is a procedure producing new chromosomes (called artificial chromosomes) from a given elite chromosome base, by using some chromosome matrix. There is some analogy with the similarity operator, however, fabrication can use more than two chromosomes from the elite base and is based on random assignment. Another difference is that we use only binary strings as chromosomes, while in [54] chromosomes as strings of symbols from a given finite set are used.

In [55] it is written that a modern evolutionary optimization method, Extreme Optimization was proposed and has since been applied to a number of combinatorial optimization problems successfully. However, Extreme Optimization has rarely been applied to continuous optimization problems. Therefore, Zeng et al. [55] have recommended the use of an Improved Real-Coded Population-Based Extreme Optimization (IRPEO) method in order to solve problems associated with unconstrained optimization. Basic IRPEO operations consist of real-coded random generation of the initial population, individual evaluation and population fitness evaluation, selection of bad elements according to the power-law probability distribution, new population generation according to the uniform random mutation, update of population through unconditional acceptance of new population. The authors have applied the IRPEO on 10 test functions with 30 dimensions, experimental results showing that IRPEO is competitive and even better compared to selected versions of Genetic Algorithm with different mutation operators. On the contrary, the algorithms presented in this thesis have been tested on 27 test functions of 2, 3, 4, 10 and 100 variables, and also on the knapsack problem.

2.4 Convergence of genetic algorithms

The concept of Evolutionary Algorithm (EA) is a collective name for those probabilistic optimization algorithms, that design is inspired by principles of biological evolution. In fact there are more similarities than differences, a general convergence theory is possible [56].

Rudolph [57] has proved, by using the Markov chain analysis, that the simple genetic algorithm, with proportional selection, crossover and mutation, converges to the optimal solution if the mutation rate is non-zero and the algorithm maintains the best solution found over time. However, as the author of [58] comment, "such a result is weak, because it is noticed that a simple random search in the space of the bit strings also converges in the same manner towards the optimum. Nothing is mentioned about the convergence speed and it can be noted that the crossover does not play any role in the result of convergence".

In [56] the author presents results that generalize the previously developed theory of convergence to arbitrary search spaces. These results are general enough to be useful for a broad class of evolutionary algorithms.

In our work we use random generation of a part of population at each iteration. This random generation gives an effect similar to mutation with non-zero rate, and also the best solution found so far is always passed to the next iteration, therefore a convergence theorem is possible (see Section 3.8).

In [59] the author discusses convergence of a general algorithm model called Random Heuristic Search (RHS). It is described by a *heuristic function* $\mathcal{G}:\wedge\rightarrow\wedge$ where $\wedge$ is a simplex in $\mathbb{R}^n$. Given the current population P_i , the next population $P_{i+1} = \tau(P_i)$ is obtained by applying some stochastic *transition rule* τ. For $p \in\wedge$, the value $\mathcal{G}(p)$ is the probability distribution that is sampled independently Γ times produce the next population.

As the author writes, " The precise definition of logarithmic time to convergence faces several obstacles. The most obvious is that ergodic random heuristic search does not converge, since it corresponds to an ergodic Markov chain. Because genetic search is typically conducted with some nonzero level of mutation, it follows that convergence, strictly speaking, does not typically take place for GAs. The naive definition of convergence as time to discover the optimal is generally useless as well. The "no free lunch theorem" (Wolpert & Macready, 1995) implies that, even with an elitest strategy and aggregating (or collapsing) all populations containing the optimal into an absorbing state, time to convergence is in general no better than that achieved by enumeration". Therefore, instead of classical convergence, the author examines conditions under which the *logarithmic convergence* holds, that is, the number of generations k required for the inequality $\|\mathcal{G}^k(p) - \omega(p)\| < \delta$ to hold is $O(-log\delta)$, where $\omega(p)$:=$\lim_k \mathcal{G}^k(p)$, $\mathcal{G}^k$ is the k-th iterate of S, and $0 < \delta < 1$. However, the obtained result requires infinite populations, a condition which is never satisfied in practical applications.

In the article [60], the authors state that genetic algorithms are widely used in solving some world optimization challenges, but few rigorous results on their convergence can be found in the literature. They show that, with a proper rigorous multistage Markov chain modeling and with simple probabilistic arguments, some convergence results for GAs can be derived. In particular, for a GA with superindividual (elitist model), the probability that the current population contains an optimal solution converges to one as the number of iteration tends to infinity. In [60], a new crossover operator is defined. It is further extended in another paper [61], where some modifications of the algorithms from [60] are introduced and their theoretical convergence is established. All these algorithms have a superindividual. Numerical comparisons among these algorithms are also included.

In [62], the authors consider a non-homogeneous genetic algorithm (NHGA) which uses two parameters (probability of mutation and probability of crossover) which can change during the execution of the algorithm. For an elitist version of this algorithm, they prove its almost sure convergence to some population containing an optimal point. By using the theory of Markov chains with finite state space, and the Chapman–Kolmogorov equation, they studied the probability of crossover and mutation for NHGA. Then the authors compare the NHGA with the homogeneous genetic algorithm (HGA). They show by some examples that there exists a non-empty subset E* of the state space that is more frequently visited when the NHGA is used. They also observe that, in the NHGA, the mutation probability should, at the beginning, be bigger that in the canonical genetic algorithm, to allow the algorithm to expand its search space. Finally, they conclude that the bigger the population size is, the closer the results for both algorithms are, but it should be noted that the computational effort increases when the size of population increases.

In [63] the authors develop sufficient conditions required for finiteness of the mean convergence time of a genetic algorithm with elitism. They also establish a lower bound for the probability of finding an optimal solution in the first m iterations. The results presented in [63] can also be extended to other optimization schemes.

In [64] the authors consider several versions of a genetic algorithm and obtain theoretical estimates for their convergence. They proved the convergence of the mean fitness of a population to the optimal value of a given function. This result is obtained for two types of GA: with crossover and mutation, and with crossover, inversion and mutation. It can also be extended to other variations of genetic operators. However, in the point of view of the authors, real-coded genetic algorithms are of special interest, but their result cannot be applied to such algorithms.

In [65] the author has obtained some stopping criteria in genetic algorithm theory, for a general model of the algorithm being a special case of the Random Heuristic Search (RHS). The approach adopted to this problem was to obtain upper bounds for the number of iterations necessary to ensure finding an optimal solution with a prescribed probability. Here “finding an optimal solution” means that the current population contains at least one copy of an individual belonging to a given set of optimal solutions.

In [66] the author studies stopping criteria for a genetic algorithm designed for solving multi-objective optimization problem. This algorithm is described in terms of a general Markov chain model. He establishes an upper bound for the number of iterations which must be executed in order to produce, with a prescribed probability, a population consisting entirely of optimal solutions. Since populations may contain multiple copies of the same element, this stopping criterion can only guarantee that at least one minimal solution is found.

In the next paper [67] the author improves the previous stopping criteria so that they enable one to find, with a prescribed probability, all minimal solutions in a finite multiobjective optimization problem.

In this thesis, we have proved the convergence of DSC algorithm in Section 3.8, this proof is suitable for any search algorithm that contains in a part of population the random generation.

2.5 Repeated runs of a genetic algorithm

In our work presented in later chapters we frequently use repeated runs of the tested algorithms to get better results. It is therefore interesting to review some theoretical considerations concerning repeated runs of genetic algorithms which are described in [68], [69].

The authors of [68] write "In given industrial application where the GA is suitable, the probability of its success cannot be too low. If the GA cannot perform with a probability of success of say 10 or 20%, then the application of GA to that particular application is risky and likely to be non-productive." However, if the algorithm is run many times on the same data, the probability of finding a good solution at least in one of the runs is of course higher. Because on this idea, the following arrangement is made in [69]: Suppose: instead of a single run of the GA, the GA algorithm runs N times, where $N > 1$, on the same data. Each run is independent, with no information passed between two runs. the best solution found is recorded after the end of each run. Then the randomly generated chromosomes are used to the begin the next run.

Suppose the probability of success of a single run is P_{SGA}. A success means that the best solution found so far is the correct (i.e. optimum) solution. Suppose the probability of success after N runs of the algorithm is P_{RGA}. It is the probability that any one of the N best solutions is the correct solution. Since the N runs are independent, we have

$$P_{RGA} = 1 - (1 - P_{SGA})^N \qquad (2.1)$$

If a user specifies a minimum acceptable value for P_{RGA} (e.g. P_{RGA} ≥0.95), the required number of N runs is simply

$$N \geq \frac{In(1 - P_{RGA})}{In(1 - P_{SGA})} \qquad (2.2)$$

Thus provided that P_{SGA} is known, one can guarantee the RGA's overall probability of success. A general technique is: Given any stochastic optimization method with probability of success P_S, by applying it N times independently, one can increase its probability of success to P_R. This is known as probability amplification or probability boasting [69].

2.6 Genetic algorithms based on schema theory

The aim of the paper [70] was to improve the search performance of the Stochastic Schemata Exploiter (SSE, already known in the literature) without sacrificing its convergence speed. For this purpose, the authors introduce the Extended Stochastic Schemata Exploiter (ESSE) and the cross-generational elitist selection SSE (cSSE). In the ESSE, once the common schemata list is defined from the common schemata which are extracted from the individuals in the sub-populations, the list is modified by deleting individual schemata, updating similar schemata, and so on. In the cSSE, a cross-generational elitist selection was introduced to the original SSE. In the numerical examples, SSE, ESSE and cSSE are compared with a genetic algorithm (GA) with Minimum Generation Gap (MGG) and the Bayesian Optimization Algorithm (BOA). Several numerical results show that the GA with MGG can find better global solutions although the convergence speed is sacrificed. In comparing the convergence speed of different algorithms, the authors notice that the cSSE and BOA are fastest among them.

In [71], a new type of multi-population GA called forking Genetic Algorithm (fGA) was suggested by Tsutsui and Fujimoto. The fGA was designed to solve multi-modal problems, which are hard to be solved by the traditional GAs since they have many local optima. The fGA algorithm was prepared to search for a single global optimum by keeping track of potential local optima. The population structure consists of a parent population and a variable number of child populations. When a certain level of similarity is detected in the parent population, the algorithm creates a child population by using the similarity calculated from the binary strings encoding (*genotypic forking*) or by using the Euclidean distance between individuals (*phenotypic forking*) to measure a phenotypic similarity. The division of the search space in genotypic forking is based on the so-called temporal and salient schemata, which detect the convergence of bit positions in the binary encoding. The child and the parent populations are not allowed to overlap.

The temporal schema reflects the population state in the current iteration, while the salient schema is calculated from the last K_h iterations. The schemata are strings consisting of the letters “0”, “1”, and “*” but the temporal schema contains a 0 or 1 if more than a predetermined percentage K_{TS} of the individuals have the same value in a gene, otherwise “*” is inserted.

The fGA is tested on two problems as test functions. One is a FM Sound’s parameter identification problem and the other is Oliver’s 30 City Travel Salesperson Problem. The results of experiments show that the fGA outperforms the standard GA.

In this thesis we use the idea of schema to find the optimum solution in a search space, in Section 3.3 we present the basics of schema theory, also we propose a free dynamic schema operator in Chapter 5.

2.7 Poor performance of the GA’s caused by defining length of schemata (messy GA)

Schemata are similarity subsets. In simple GAs, schemata may be represented by the usual similarity template notation, where a wildcard character (usually a *) is used to indicate positional indifference. In messy GAs, genes are allowed to change position, and in the messy coding, the ordering of a gene does not directly affect its allele's fitness [72].

The distance between the leftmost and rightmost 1 in a bit string is called the defining length of this string. For example, the defining length of string 00100110 is 4, and defining length of string 10000001 is 7 [72].

As the authors of [73] write (Section 6.3), the reason of poor performance of the classical GA on some specially constructed “deceptive” functions is that the useful schemata (i.e. the ones that lead to good solutions) have too large defining lengths.

Consequently, the building blocks for an optimal solution are easily destroyed by crossover.

The following example taken from [72] explains this problem: “For example, suppose the schema 00**** is highly fit and the schema ****00 is highly fit , but the schema 00***00 is much less fit than its complement, 11***11, which itself is a building block of the optimal point, 1111111. In the particular case, the GA will tend to converge to points representative of the less fit schema (perhaps points like 0011100), because with high probability, crossover will tend to disrupt the needed combination (11***11)”.

In our algorithms we overcome this difficulty by using the dissimilarity operator, this operator can change 0’s in the schema 00***00 to 1’s by testing the dissimilarity between the current chromosome and the second one, then generating randomly 0 or 1 [see Chapter 3].

2.8 Performance measures [53]

The quality assessment of an evolutionary algorithm usually involves empirical comparisons between the given EA and other algorithms. also the parameter tuning for good performance requires some experimental work to compare different versions of the same algorithm. Since some parameters of EAs are random, performance measures have statistical nature, which means that a number of experiments must be performed to obtain sufficient experimental data.

There are three basic performance criteria:

• Success Rate (SR)

• Effectiveness (solution quality)

• Efficiency (speed)

The **Success Rate** (SR) can be defined as the percentage of runs in which success occurs, where success mean finding a solution with desired quality. However, it is difficult or impossible to use for problems where the optimum solution cannot be identified.

The **Mean Best Fitness Measure** (MBF) can be defined for any problem that is dealt with by an EA. Suppose that, is a measure of effectiveness that at the end of each run, the EA records the best fitness obtained. The MBF is the average of these values for all runs.

Talking about the algorithm **efficiency** or speed, it is often measured in elapsed computer time or user time. However, these measures depend on the hardware, operating system, compiler, and so on. In other words, repeating the same experiments, elsewhere, may yield different results.

It is always measured on a number of independent runs. Therefore the **Average number of Evaluations to a Solution (AES)** is used as a measure of efficiency, "Sometimes the average number of evaluations to termination measure is used instead of the AES, but this has clear disadvantages. Namely, for runs finding no solutions, the specified maximum number of evaluations will be used when calculating this average. This means that the values obtained will depend on how long the unsuccessful runs are allowed to continue. That is, this measure mixes the AES and the SR measures, and the outcome figures are hard to interpret".

In our work we have used the SR, MBF and AES measures.

2.9 Initial population effects

According to [74], the initial population is important in an evolutionary algorithm, since it affects the speed of convergence and the final answer quality. In case there is no available information about a solution, random initialization is applied as a method to produce the candidate solutions for the initial population.

In [74] a novel initialization of the population is proposed that uses opposition-based learning to generate initial populations which can be used instead of a purely random initialization. Through the conducted experiments it is demonstrated that when an opposition-based population replaces random initialization, the convergence speed is accelerated. Thus it is proposed that opposition-based approach should be used in the optimization of a population initialization. The multimodal and unimodal test functions are used to verify the experiment. The experiment results record the average convergence speed 10% faster. According to the proposed algorithm, it is recommended that one should start with an appropriate population in cases where there is no information related to a solution. The authors have applied their idea to Differential Evolution, but it is also applicable to other population-based optimization algorithms, for instance, the genetic algorithms that form a future direction of the authors' work.

The authors of [75] conducted some initial population difference tests for the real coded genetic algorithms. Whereas the genetic algorithms are commonly used metaheuristics for global optimization, very little research has been done on the generation of initial populations. In [75] authors search for an answer to the question what is the effect of initial populations. Also, does the initial population play a role in the performance of a genetic algorithm, and if so, how it should be generated? They study the characteristics of different point generators, using four main criteria: ”the uniform coverage and the genetic diversity of the points as well as the speed and the usability of the generator”. With a simple academic example, the authors show that initial population has a significant effect on the best objective functional value over several generations. Then they focus on studying different methods of generating an initial population for the case without a priori information on the location of the global minima.

In [76] the authors present a systematic review of the existing population initialization techniques. They categorize these techniques according to three criteria: randomization, compositionality and generality. Each criterion leads to some division of the methods into several sub-categories. The authors stress that the area of population initialization methods was one of the least explored in evolutionary algorithms.

There is a common step in all evolutionary algorithms - it is a population initialization. The role of this step is to provide an initial guess of solutions. Then, subsequently, these planned solutions will be improved in the process of optimization until the stop criterion is met. If these initially guessed solutions are good, the EA can find the optimum solution quickly, otherwise, the EA can be prevented from finding the optimum solution.

In our work, we have used a big initial population in the last algorithm (IPMFDS) in Chapter 6. It shows better results comparing with other our algorithms and also with CMA-ES and DE for most tested functions.

In [77] the authors proposed genetic algorithm with variable population size (GAVaPS) This method depends on the concept of age and life of the individual. When an individual is created, either during the first generation or through the variation operator, it has age zero. Then, for each generation the individual survives, his age increases by 1. At birth, the lifetime of each individual is determined and corresponds to the number of generations in which the individual survives in the population. When the age of the individual exceeds the lifetime, the individual dies and is disposed of. In each generation, a certain fraction of the current population is allowed to regenerate. Each individual has an equal probability of being selected for reproduction. The selection is achieved indirectly by utilizing the lifetime that is assigned to individuals. Those with higher than average fitness have a greater lifetime than those with less than average fitness. The idea is that the better the individual is, the more it should be allowed to stay in the population, and thus propagate its traits to future individuals.

2.10 Modified genetic algorithms

In [23] the authors developed an effective new technology to improve the speed of convergence of a genetic optimization algorithm. They applied this modification of the GA to chemical engineering problem. They have investigated and provided a number of sampling techniques to create a good initial populations that encourages exploration through the search space. These sampling techniques include “Latin hypercube sampling (LHS), Faure sequence sampling (FSS), and Hamersley sequence sampling (HSS)”, these samples are used to select a good first population group. The performance of the proposed algorithm is compared with an algorithm having the random initial population in terms of solution quality and speed of convergence. Their technology provides a better solution and their algorithm converges to the global optimal solution faster than the classical GA.

In [78] the author pointed that the best chromosome is not always improved in every generation in the simple genetic algorithm. Good solutions obtained, can be destroyed by crossover or mutation or both of them. The modified GA aims is to avoid this disadvantage by changing order of genetic operations: selection now appears after crossover and mutation. This algorithm has been proposed to determine a parameter for the E. coli fed-batch fermentation model. The use of the proposed modified GA for a parameter identification of fermentation processes is highly efficient and effective which is illustrated by the simulation results.

In [79] the authors propose a new Genetic Algorithm (GA) to optimize multimodal continuous functions, this method uses a genetic algorithm with real-value coding (RCGA) and applies several existing techniques such as the real coding and the composition of sub-populations based on the entropy theory. The idea of RCGA is based on careful balance between both tasks usual in heuristic search: "intensification" and "diversification". The authors divide the classical GA into three processes. The first process creates several appropriate subpopulations by using the theory of information entropy. The second process applies genetic operators to each subpopulation to gradually enrich it with better individuals. The last process determines the best point among the best solutions issued by each of the previous subpopulations. Then, in some neighborhood of this point, a new population is generated for a traditional GA. In this way, the population is fully renovated after each generation. The size of the neighborhood is reduced after each generation. A comparison of performances with several stochastic global search methods is included, using some test functions. The technique is advisable to solve highly multimodal problems.

In [80] the authors suggest a modified method based on GA called Box Complex (BC), which has developed from the Simplex method of optimization. This method gives gradual convergence with small population size, and it has also some ability to escape from getting trapped in local minima. To avoid the big computational effort with bigger population, the authors suggest to integrate the convergence feature of Box Complex method with global search feature of GA. At every generation, they add new member(s) to the population, by replacing the equal number of the most inferior member(s). Hence the population size is constant.

2.11 Hybrid algorithms

In [49] the author introduced a hybrid genetic algorithm, consisting of Genetic Algorithm (GA), heuristics and Ant Colony Optimization (ACO). It was proposed to solve Split Delivery Vehicle Routing Problem (SDVRP) and was tested this problem. Due to the constraints of a SDVRP, it is not possible to directly use classical GA for this problem to obtain a feasible set of offspring. A modification of crossover is necessary, or another possibility is to remove infeasible solutions after mutation and replace them with the solutions having higher fitness value in the old population. Briefly, the hybrid algorithm generates and evaluates a big initial population (1000) by using ACO, then it choses 500 routes of the best solutions, then puts them in the modified genetic algorithm to form an initial generation. A single iteration of the modified GA chooses the best 5 routes of the previous generation and adds them to the future generation (elitism), then chooses 2 parents randomly from the previous generation and performs a one-point crossover, then applies the heuristics to build new routes and adds them to the future generation; this procedure is repeated until 50 population members are created. Then the algorithm evaluate the fitness of the future generation and sorts it according to the shortest distance. This is repeated for 100 iterations of the modified GA to get results. The hybrid GA shows the ability to provide better results and faster computational time for the datasets the author’s study.

In our work we have applied a similar idea of big initial population in IPMFDS. First, the initial population is evaluated, then the best solutions are taken from it and inserted as the first generation to the original MFDS algorithm. We use 500, 1000 and 2000 elements in the initial population for 2-, 10- and 100-dimensional functions respectively.

The authors of [32] say that the field of mobile robotics, the global path planning is a challenging problem because of its complexity and nature which is nondeterministic polynomial-time hard (NP-hard). To solve this problem, they suggested a new hybrid optimization algorithm by developing the PSO and DE algorithms, then integrating them. The developed PSO is called Nonlinear Time-Varying PSO (NTVPSO) to update the positions of particles velocities in the hybrid algorithm, trying to avoid stagnation. The updated DE named the Ranking Based Self-Adaptive DE (RBSADE), is enhanced to include the personal best experience of particles in the hybrid algorithm. Whereas particle swarm optimization considered most popular in global path planning because of its high convergence speed and simplicity, on the other hand, the basic PSO has problems with balancing exploration and exploitation, and also suffers from recession, hence its efficiency may be restricted in solving global path planning. The authors of [32] named this hybrid algorithm HNTVPSO-RBSADE, which integrates NTVPSO with RBSADE. At first the particles depend on moving rules in NTVPSO to change their positions and velocities. Then the RBSADE algorithm is enhanced to include the best positions of particles to avoid stagnant. On four numerical simulations and a Monte-Carlo experiment this algorithm is tested against four evolutionary algorithms: Adaptive Differential Evolution (JADE), Time- Varying Particle Swarm Optimization (TVPSO), Gravitational Search (GS) and modified Genetic Algorithm (mGA), and outperforms the other four algorithms.

In [36] two famous algorithms: Biogeography Based Optimization (BBO) and Artificial Bee Colony (ABC) are used to form a hybrid algorithm called (HBBABC). It utilizes the exploitation features of BBO and exploration features of ABC. This hybrid algorithm was tested on 14 benchmark problems to confirm its performance taking into account discrete design variables and 5 engineering design optimization problems. Different criteria are also taken into account like Mean Solution, Best Solution, T-test, Success Rate and other criteria. Overall performance of HBBABC is better than BBO and ABC in experimental results with using the same criteria above.

In [33] the author proposes a new real-coded evolutionary algorithm to apply on path synthesis of a four-bar linkage. In this new evolutionary algorithm the author combines Differential Evolution (DE) with the Real-valued Genetic Algorithm (RGA). This hybrid algorithm is called “GA–DE hybrid algorithm.” The content of the crossover is the only difference between the proposed algorithm and RGA: the author replaces the crossover operation in the RGA with differential vector perturbation, with the best individual or some excellent individuals as the base vectors. This method was tested on four cases which showed that more accurate solutions were obtained for three cases than those gained by other evolutionary methods.

2.12 Using binary encoding in real-valued function optimization

Arabas [81] in Section 4.11 poses the question if it is worth using binary encoding in EAs for solving numerical optimization problems. After analyzing several examples, he concludes that this method is not advisable because it introduces serious perturbations into the search process. The reason is that the distance in the space of binary string (the space of genotypes) is different from the distance in $\mathbb{R}^n$ (the space of phenotypes). Consequently, two chromosomes which are close to each other as binary strings, after decoding may be positioned far from each other in $\mathbb{R}^n$, and vice versa. The author presents the following example of an irregular behavior of the binary crossover operator in $\mathbb{R}^2$: assuming that we have two parents $(x_1, y_1) = (4, 6) = (0100, 0110)$ and $(x_2, y_2) = (7, 9) = (0111, 1001)$, denoted as black circles, the following points can be reached by one-point crossover (white circles):

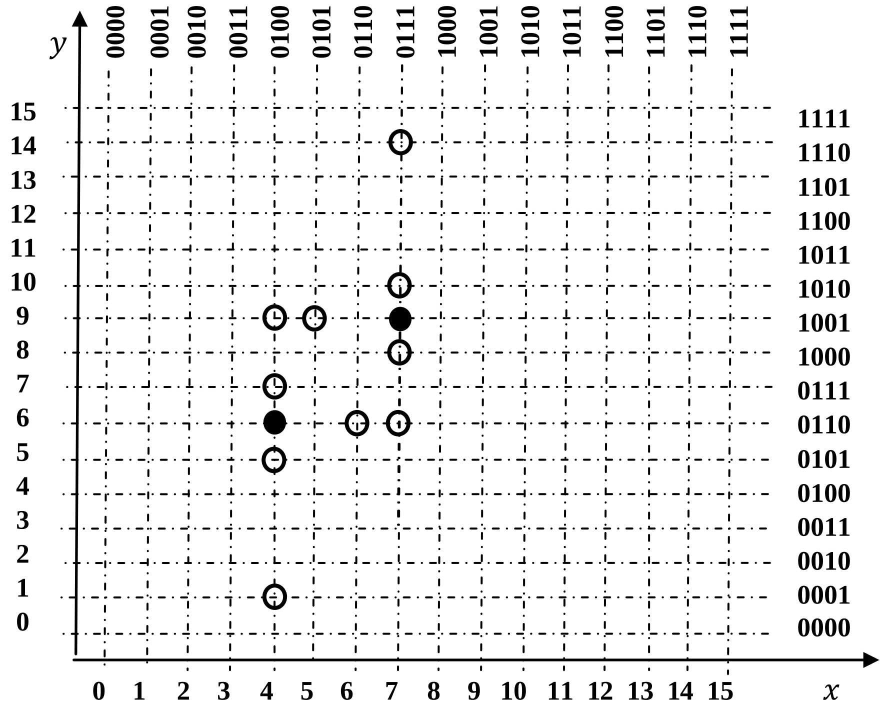


Figure 2. 1 The set of chromosomes available by applying one-point crossover (source: [81], Figure 4.17)

We see that some point lying "between" the two parents cannot be reached, while some other points far from the parents can.

In this context we would like to analyze the behavior of two operators introduced in this thesis in Chapter 3: the similarity and the dissimilarity operator. Considering the same example, we now have the following two sets of points:

- green squares - points that can be reached by the similarity operator,
- red stars – points that can be reached by the dissimilarity operator:

Similarity

P1	0	1	0	0	0	1	1	0
P2	0	1	1	1	1	0	0	1

Dissimilarity

P1	0	1	0	0	0	1	1	0
P2	0	1	1	1	1	0	0	1

Result of Similarity operator

G.P	0	1	*	*	*	*	*	*

Result of Dissimilarity operator

R.P	*	*	1	1	1	0	0	1

Where G.P = Green Points, R.P = Red Points

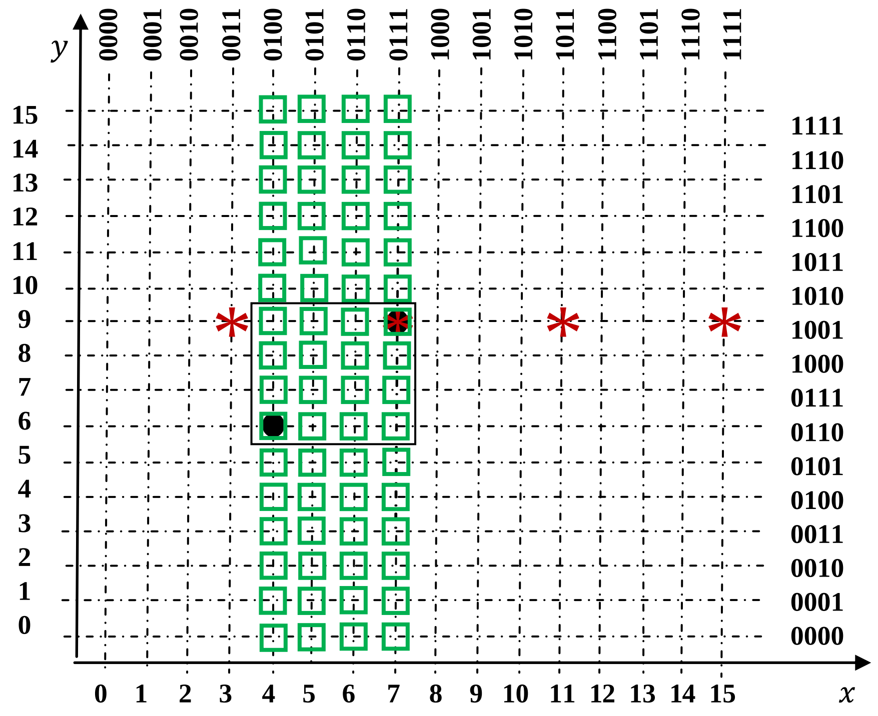


Figure 2. 2 The set of chromosomes available by applying the similarity and the dissimilarity operators

We can see from this picture that some of the generated points are really far from the parents. However, contrary to Figure 2.1, now all the points lying "between" the parents (that is, points of the square determined by them) are covered by the chromosomes generated by our operators. Note that our algorithms do not use the classical one-point crossover that has irregular behavior presented on Figure 2.1.

In our algorithms we have chosen the binary representation instead of the real-value representation because binary representation can be used both for real problems and binary problems (like the knapsack problem). The binary encoding is especially important in our similarity and dissimilarity operators (see Chapter 3) which have high probability to find new better solutions near to the existing points in a population, as shown in Figure 2.2. Also, the schema theory is used another way in the dynamic schema and dynamic free schema operators (see Chapters 4 and 5) to specify most suitable chromosomes for the optimum solution.

CHAPTER THREE: The DSC Algorithm

3.1 Introduction

In this chapter, a new evolutionary optimization algorithm is described which explores similarities and dissimilarities in pairs of chromosomes. This procedure divides each population into three not equal parts, and then applies new genetic operators to the first two of them. Our algorithm is called Dissimilarity and Similarity of Chromosomes (DSC) and its purpose is to find optimal solutions in numerical optimization problems.

For the construction of the two genetic operators used here – the dissimilarity operator and the similarity operator – the notion of a schema plays an important role. The explanation of the idea of a schema is given in Section 3.3.

To demonstrate the performance of the DSC algorithm, it is run on 18 two-dimensional, one four-dimensional and five ten-dimensional optimization problems described in the literature, and compared with the well-known GA, Covariance Matrix Adaptation Evolution Strategy (CMA-ES) and Differential Evolution (DE) algorithms. The results of tests show the superiority of our strategy in the majority of cases.

The concept of dividing a population into parts and then working with schemata and similarity for each part separately, is already known in the literature. For example, in the paper by Han et al. [82] the population was divided into three parts based on the fitness of chromosomes (the best, the middle and the worst fitness groups) and then the common schema in a population was discovered by using clustering. Later, for the first and the third part of a population, the number of chromosomes that have some similarity with the schema was calculated. The percentage of positions on which the individual agrees with the schema defines the similarity between an individual and a schema.

A general approach to estimate the expected first hitting time (i.e., the time when the algorithm finds an optimal solution) was proposed by Yu and Zhou [83]. It is based on analysis of EAs with different configurations. This method works with three mutation operators, a recombination operator and a time variant mutation operator. We are planning to examine the possibility of applying a similar theoretical analysis to our DSC algorithm in further research.

In this chapter we present both theoretical and experimental results on the new DSC algorithm. The chapter is organized as follows. In Section 3.2, we introduce two genetic operators that are used in the DSC: the similarity operator and the dissimilarity operator. They are precisely defined in Section 3.4 in terms of forma analysis of Radcliffe [84]. Section 3.3 presents a basics of schema theory. Section 3.5 gives the analysis of experimental results. Section 3.6 contains the discussion of figures. Section 3.7 contains some information about the parameters setting in GA used for comparison with our algorithms. In Section 3.8, convergence of DSC is presented. Finally, some conclusions for the DSC are mentioned in Section 3.9.

3.2 The idea of similarirty and dissimilarity operators

In this section, we explain by a simple example how our two genetic operators could help in obtaning better solutions.

Suppose f is a one-dimensional function with the domain [0,1], as shown in Fig. 3.1. This domain is represented by binary representation consisting of four bits, (0000,0001,…,1111), that means the range is divided into 16 segments.

The principle of similarity operator is as follows: Suppose there are two best solutions in a population : 0010 and 1011, colored in gray. If the similarity operator is applied (see Table 3.2), and bits number 1 and 4 for each chromsome are not the same, then we put * in the second chrmosomes instead of them, and then randomlly put 0 or 1 in positions having *s. Thus, a better solution is possible to be found, as shown in the green part in Figure 3. 1.

Bits:	1234
Ch.1:	0010
Ch.2:	1011
Ch.2:	*01*
Ch.2:	1010

Now the principle of dissimilarity operator: Suppose there are the same chromosomes in gray color. If the dissimilarity operator is applied, and bits number 2 and 3 for each chromsome are the same, then we put * in the second chrmosomes instead of them, and then randomlly put 0 or 1 in positions having *s. Thus, a better solution is possible to be found, as shown in the red part in Figure 3. 1.

Bits:	1234
Ch.1:	0010
Ch.2:	1011
Ch.2:	1**1
Ch.2:	1101

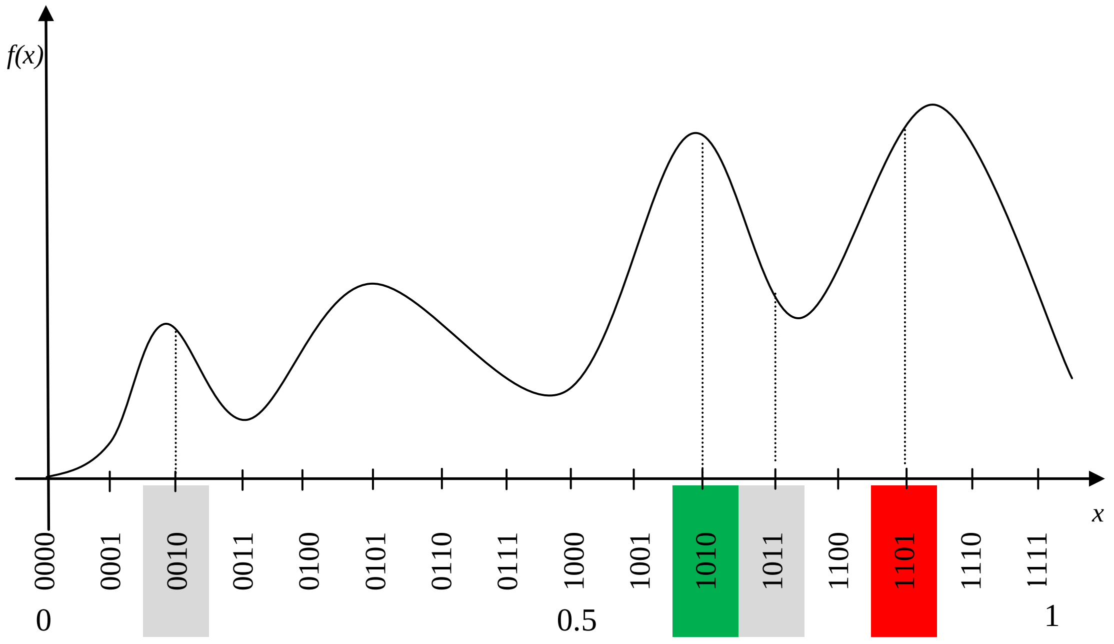


Figure 3. 1 A simulation of similarity and dissimilarity idea.

3.3 The basics of schema theory

In this section, some base for the concept of a schema is provided. A schema is a template that gives the representation of a set of solutions of genetic algorithms. In binary coding, a schema is usually presented as a string of symbols from the alphabet { 0, 1, * }, where the character * can be interpreted as a "0 or 1 is all ok". For example, the schema 01*00* represents 4 chromosomes: 010000, 011000, 010001 and 011001. Generally, a schema is a frame for groups of chromosomes that have the same fixed sections [85].

Definition 3.3 in [85], says: "according to the schema theorem, under the operation of the genetic operators such as selection, mutation and crossover, the schema with a low order, short defining length and its average fitness higher than the population average fitness will increase exponentially in the offspring". A schema that involves less locations with *s is more specific than a schema with more locations with *s [86].

Note that it is not true that every subset of the set of bit strings of length L can be described as a schema; in fact, the vast majority cannot. There are 2^{L} possible bit strings of length L, and thus $2^{2^{L}}$ possible subsets of strings, but there are only 3^{L} possible schemas. However, a central assumption of the traditional GA theory is that schemas are in fact the building blocks that the GA processes effectively under the operators of selection, mutation, and single-point crossover [14].

3.4 Forma analysis of genetic operators

In this section, we define and analyze two genetic operators used in our DSC algorithm.

We apply the abstract forma analysis presented in [84], so that our definitions may be applied in a more general setting than only for binary schemata. First, we must review some definitions.

Let S be a finite search space of some genetic algorithm. A function $\psi : S \times S \longrightarrow \{0,1\}$ is called an *equivalence relation over* S if and only if it satisfies the following three conditions:

1. $\forall x \in S : \psi(x,x) = 1$,
2. $\forall x,y \in S : \psi(x,y) = 1 \Rightarrow \psi(y,x) = 1$,
3. $\forall x,y,z \in S : \psi(x,y) = \psi(y,z) = 1 \Rightarrow \psi(x,z) = 1$.

We denote by $\mathbb{E}(S)$ the set of all equivalence relations over S. Given two equivalence relations $\psi, \varphi \in \mathbb{E}(S)$, we define their *intersection* $\psi \cap \varphi \in \mathbb{E}(S)$ by

$$(\psi \cap \varphi)(x,y) := \psi(x,y) \wedge \varphi(x,y),$$

where $\wedge$ denotes logical conjunction ("and").

For a given set $\Psi \subset \mathbb{E}(S)$, we call a subset $E \subset \Psi$ a *basis* for Ψ if and only if the following two conditions hold:

1. E *spans* Ψ, that is, every element of Ψ can be constructed by intersection of some subset of E:

   $$\Psi \subset \text{Span } E := \{\mathcal{E} \in \mathbb{E}(S) : \exists A_{\mathcal{E}} \subset E \text{ such that } \cap A_{\mathcal{E}} = \mathcal{E}\}.$$

2. E is *independent*, that is, no member of E can be constructed by intersection of other members of E:

   $$\forall \mathcal{E} \in E, \nexists A_{\mathcal{E}} \subset E \setminus \{\mathcal{E}\} : \cap A_{\mathcal{E}} = \mathcal{E}.$$

Given an equivalence relation $\psi \in \mathbb{E}(S)$, we define Ξ_{ψ} to be the set of *formae* (equivalence classes) induced by ψ . Further, given a set of equivalence relations $\Psi \subset \mathbb{E}(S)$, with $\Psi = \{\psi_1, \psi_2, \dots, \psi_{|\Psi|}\}$, where $|\Psi|$ is the number of elements of Ψ, we define Ξ_{Ψ} to be the set of vectors of formae given as the Cartesian product

$$\Xi_{\Psi} := \Xi_{\psi_1} \times \Xi_{\psi_1} \times \dots \times \Xi_{\psi_{|\Psi|}}.$$

A set of equivalence relations $\Psi \subset \mathbb{E}(S)$ is said to *cover* S if and only if for each pair of different solutions in S there exists some relation in Ψ under which the pair are not equivalent:

$$\forall x \in S, \forall y \in S \setminus \{x\}, \exists\, \psi \in \Psi : \psi(x,y) = 0.$$

Let E be a basis for a set of equivalence relations $\Psi \subset \mathbb{E}(S)$ that covers S. The members of E are called *basic equivalence relations*, or *genes*. For a given relation $\mathcal{E} \in E$, the members of $\Xi_{\mathcal{E}}$ are called *basic formae*, or *alleles*.

A set of equivalence relations $E \subset \mathbb{E}(S)$ is said to be *orthogonal* if and only if, given any $|E|$ equivalence classes induced by different members of E, their intersection is nonempty:

$$\forall \xi = (\xi_1, \xi_2, \dots, \xi_{|E|}) \in \Xi_E : \bigcap_{i=1}^{|E|} \xi_i \neq \varphi$$

Let Ξ be a set of formae defined over a search space S, and let $L \subset S$. The *similarity set* of L (defined with respect to Ξ and written $\sum(L)$) is the intersection of all those formae to which each solution in L belongs:

$$\sum(L) := \begin{cases} \cap \{\xi \in \Xi : L \subset \xi\}, if\ \exists \xi \in \Xi : \mathrm{L} \subset \xi, \\ S, \qquad\qquad\qquad\qquad \text{otherwise.} \end{cases}$$

For a given set $E = \{\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_n\} \subset \mathbb{E}(S)$, we define the *genetic representation function* $\rho_E(x): S \to \Xi_E$ by

$$\rho_E(x) := ([x]_{\mathcal{E}_1}, [x]_{\mathcal{E}_2}, \dots, [x]_{\mathcal{E}_n}),$$

where, for given $\mathcal{E} \subset \mathbb{E}(S)$ and $x \in S$, we denote by $[x]_{\mathcal{E}}$ the equivalence class of x under $\mathcal{E}$:

$$[x]_{\mathcal{E}} := \{y \in S : \mathcal{E}(x, y) = 1\}.$$

Now, we are able to define the two genetic operators used in our DSC algorithm. The first one, the similarity operator, can be defined without any extra assumption on the considered set Ψ of equivalence relations. It is in fact equal to the *random respectful recombination operator* $\mathbf{R^3}: S \times S \times \mathbb{Z} \to S$ ([84], Def. 59) defined by

$$\mathbf{R^3}(x, y, k) := \sigma_{k'}(x, y),$$

where $\mathbb{Z}$ is the set of integers, $\sigma_i(x, y)$ is the ith element of the similarity set $\sum(\{x, y\})$ under some arbitrary enumeration, and $k' := k\ (\mathrm{mod}|\sum(\{x, y\})|)$. The number

k is interpreted as a random control parameter; thus $\mathbf{R^3}(x, y, k)$ returns a randomly selected element of the similarity set of x and y. The *similarity operator* is defined as

$$\mathrm{sim}(x, y, k) \coloneqq \mathbf{R^3}(x, y, k).$$

The second operator, the dissimilarity operator, is defined under the additional assumption that an orthogonal basis $E = \{\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_n\}$ for Ψ is given that covers S. Then it follows from ([84], Thm. 25) that ρ_E is a bijection. Moreover, we assume that each basic relation $\mathcal{E} \in E$ divides the search space S into two equivalence classes (i.e. ,for each gene, there are only two alleles available). For each $x \in S$, we can thus define the c*omplement* of the class $[x]_{\mathcal{E}}$, denoted by $\overline{[x]_{\mathcal{E}}}$, as follows:

$$\overline{[x]_{\mathcal{E}}} \coloneqq \{y \in S \colon \mathcal{E}(x, y) = 0\}.$$

Of course, $\overline{[x]_{\mathcal{E}}}$ is also some equivalence class under $\mathcal{E}$. Since ρ_E is bijective, we can also define the *opposite element* to x, denoted $\bar{x}$, as follows:

$$\bar{x} \coloneqq \rho_E^{-1}\left(\overline{[x]_{\mathcal{E}_1}}, \overline{[x]_{\mathcal{E}_2}}, \dots \overline{[x]_{\mathcal{E}_3}}\right).$$

Then we define the *dissimilarity operator* (depending on two elements $x, y \in S$ and a random control parameter $k \in \mathbb{Z}$) by

$$\mathrm{dis}(x, y, k) \coloneqq \mathrm{sim}(\bar{x}, y, k).$$

It follows from the theory presented in [84] that the similarity operator possesses some properties required by a “good” recombination (crossover) operator. In particular, it respects the formae with respect to which it is defined, in the sense that we always have $\mathrm{sim}(x, y, k) \in \sum(\{x, y\})$. On the other hand, the dissimilarity operator does not have such properties; it is a composition of the similarity operator and the operation of taking the opposite of the first argument.

In our DSC algorithm, the chromosomes (i.e., the values of ρ_E) are simply binary strings of a fixed length, and the basic equivalence relations in E are determined by fixed positions in a string (i.e., two strings are equivalent if they have the same value at a given position). Then the equivalence relations from Span E are the usual schemata (each schema is determined by a finite number of fixed positions in a string). In this particular case, the similarity operator is equivalent to the well-known *uniform crossover* (see [84],

p. 370), while the dissimilarity operator is equivalent to the uniform crossover applied to $\bar{x}$ and y.

3.4.1 The DSC algorithm

The following optimization problem is considered:

$$f: \mathbb{R}^n \longrightarrow \mathbb{R}$$

$$\text{minimize}|\text{maximize } f(x_1, \dots, x_n) \quad \text{subject to}$$

$$x_i \in [a_i, b_i], \qquad i = 1, \dots, n$$

where $f: \mathbb{R}^n \longrightarrow \mathbb{R}$ is a given function.

In the algorithm described below, we use a standard encoding of chromosomes as in the book of Michalewicz [87]. In particular, it uses the following formula to decode a real number $x_i \in [a_i, b_i]$:

$$x_i = a_i + \text{decimal}(1001..001) * \frac{b_i - a_i}{2^{m_i} - 1}$$

where m_i is the length of a binary string and "decimal" represents the decimal value of this string. The value of m_i for each variable depends on the length of the interval $[a_i, b_i]$. To encode a point $(x_1, \dots, x_n)$, a decimal string of length $m = \sum_{i=1}^{n} m_i$ is used.

Let M be a positive integer divisible by 8. The DSC algorithm consists of the following steps:

1. Generate M chromosomes, each chromosome representing a point $(x_1, \dots, x_n)$.
2. Compute the values of the fitness function f for each chromosome in the population.
3. Sort the chromosomes according to the descending (for maximization) or ascending (for minimization) values of the fitness function, divide the population into three not equal groups: G1 is the first quarter, G2 is the second quarter and G3 is the second half of population.
4. Copy C times the first chromosome and put it in C positions in the first half of the population randomly, replacing the original chromosomes, where $C = M/8$.

5. Compare pairs of chromosomes for the first half of the population to find dissimilarities and similarities. Check each two following chromosomes, i.e. the first and the second, the second and the third, and so on, by comparing the respective bits, as follows:

   (a) For chromosomes in G1 (from 1 to $M/4$), if the two bits are equal, put a star (*) in the second (following) chromosome; otherwise leave this bit without change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one, and so on.

Table 3. 1 The dissimilarity operator.

Before change: example for the first quarter of chromosomes

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	0	1	1	0	0	0	1

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	*	0	1	1	0	*	0	*

After change: put randomly 0 or 1 in (*) bits

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	0	1	1	0	0	0	0

   (b) For chromosomes in G2 (from $M/4 + 1$ to $M/2$), if the two bits are not equal, put a star (*) in the second (following) chromosome; otherwise leave this bit without change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one, and so on.

Table 3. 2 The similarity operator.

Before change: example for second quarter of chromosomes

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	0	1	1	0	0	0	1

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	*	*	*	*	0	*	1

After change: put randomly 0 or 1 in (*) bits

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	1	0	1	0	0	1	1

6. All chromosomes B created this way replace the original ones on positions from 2 to $M/2$. Then generate randomly chromosomes for G3. These will replace the second half of the chromosomes (on positions from $M/2 + 1$ to M).
7. Go to step 2 and repeat until the stopping criterion is reached.

Notes.

(a) The genetic operator performing the operations shown in Table 3. 1 on a pair of chromosomes A and B is called the *dissimilarity operator*, and the genetic operator performing the operations shown in

Table 3. 2 is called the *similarity operator*.

(b) The stopping criterion for the algorithm depends on the example being considered, see Section 3.5.

To maintain population diversity, Sultan et al. [88] proposed a simple injection strategy to the population. They use fix point injection, which means that they introduce new randomly generated chromosomes to the population for certain numbers of generations. A similar strategy in the DSC algorithm has been applied by generating the second half of each population randomly.

Figure 3. 2 presents the flowchart of the DSC algorithm.

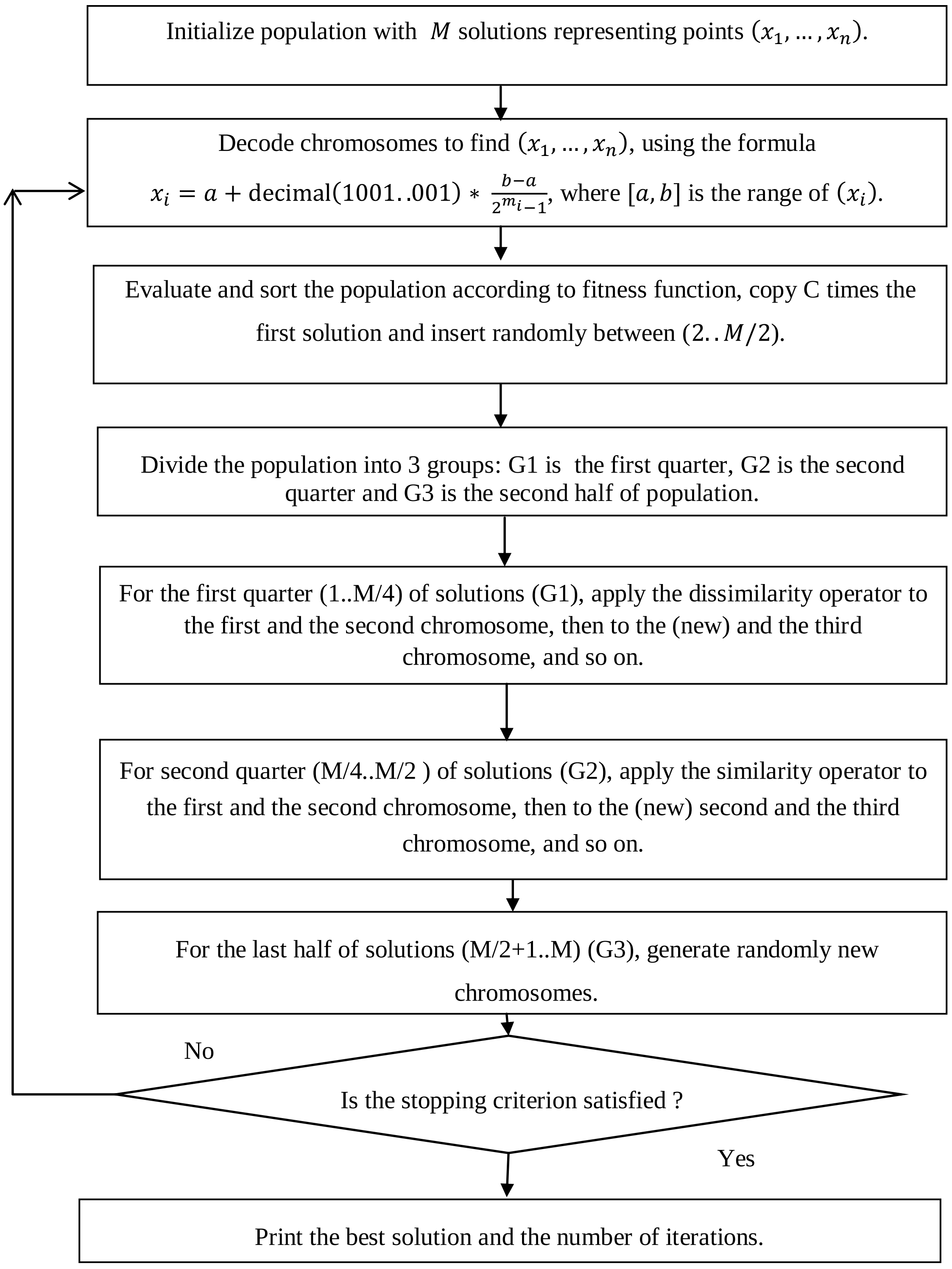

Initialize population with M solutions representing points $(x_1, \dots, x_n)$.
Decode chromosomes to find $(x_1, \dots, x_n)$, using the formula $x_i = a + \text{decimal}(1001..001) * \frac{b-a}{2^{m_i-1}}$, where $[a,b]$ is the range of (x_i).
Evaluate and sort the population according to fitness function, copy C times the first solution and insert randomly between $(2..M/2)$.
Divide the population into 3 groups: G1 is the first quarter, G2 is the second quarter and G3 is the second half of population.
For the first quarter (1..M/4) of solutions (G1), apply the dissimilarity operator to the first and the second chromosome, then to the (new) and the third chromosome, and so on.
For second quarter (M/4..M/2) of solutions (G2), apply the similarity operator to the first and the second chromosome, then to the (new) second and the third chromosome, and so on.
For the last half of solutions (M/2+1..M) (G3), generate randomly new chromosomes.
No
Is the stopping criterion satisfied ?
Yes
Print the best solution and the number of iterations.

Figure 3. 2 Flowchart of the DSC algorithm.

In the paper by M. Lewchuk [89], the author introduces a genetic invariance algorithm which is a modification of the classical GA. He uses a uniform crossover operator with is equivalent to our similarity operator, and he also uses sorting of the population according to the fitness function values. However, the crossover is applied only to a pair of individuals for which the difference in their function values is minimum over all pairs. Note that the uniform crossover and the sorting procedure is used in our DSC algorithm, but we also use a new dissimilarity operator and random regeneration of a part of population in each iteration; these last two procedures do not appear in the genetic invariance algorithm.

In Berretta et al. [90] the authors define the Recombine() procedure (pp. 78-79) which contains three genetic operators called “rebel”, “conciliator” and “obsequent”. They take some alleles from two parents P_1 and P_2 to copy in the offspring first as follows:

1. “rebel” copies alleles of P_2 which are different from P_1,
2. “conciliator” copies alleles in common to P_1 and P_2,
3. “obsequent” copies alleles of P_1 which are different from P_2.

Then the procedure chooses the alleles for the remaining positions in the offspring. This can be done by using several different algorithms (random or deterministic). It should be noted that the “rebel" operator is very similar to our dissimilarity operator, in fact, there are equivalent if a random selection is chosen for the second part of the procedure. In the same way, the “conciliator” is equivalent to our similarity operator, and “obsequent” is equivalent to our dissimilarity operator applied to P_2 and P_1 (in reverse order).

3.5 Experimental results

In this section, we report on computational testing (by using the Matlab software) of the DSC algorithm on 22 test functions taken from literature (Appendix A). After each test, the result of DSC has been compared with the known global optimum and with the result of a classical GA taken from our experiments (see Table 3.10), also, in Table 3.7 a comparison of the mean number of function evaluations and success rate of CMA-ES, DE and DSC algorithms presented.

The results are presented in Table 3.3–3.6 below. We have applied the algorithm with 40 chromosomes (see the results in table 3.3), 80 chromosomes (Table 3.4) and 160 chromosomes (Table 3.5). The DSC algorithm has found optimum solutions for some optimization prob- lems (Schwefel's) that the classical genetic algorithm cannot solve, with the minimum success rate 92% with 80 chromosomes for Schwefel's function (Table 3.4) and the maximum success rate 100% for the remaining problems. Observe that with 160 chromosomes we have got 100% success rate even for the Schwefel's example. On the other hand, for 10-dimensional problems, the success rate for the DSC is worse than GA, we use the following parameters for GA (population type is bit string, 200 chromosome, two point crossover, 2500 iterations as maximum), see Table 3.9.

In Table 3. 6 we compare the mean number of iterations for all successful runs of the proposed DSC (40, 80 and 160 chromosomes). Then we compare the rates of success of the DSC and the classical GA algorithms. The algorithm was stopped when either the maximum number of iterations (fixed to 2500) was reached or the difference between the obtained minimum/maximum fitness and the global optimum was less than or equal to the threshold given in the second column.

The success rates for the GA presented for comparison in the last columns of

Table 3. 3-3.7 were taken from the best results of our experimental work (Bit string or Double vector for the population type); these results were obtained for populations 80 chromosomes, 2500 iterations, two point crossover, see Table 3. 10 for more details.

Table 3. 7 presents the comparison of CMA-ES, DE and DSC algorithms in terms of mean number of function evaluations and success rate for 50 runs, maximum number of iterations 2500, with population size equal to 80 chromosomes.

We have recognized that, for most problems, using 80 chromosomes gives the best results in terms of both success rate and the number of function evaluations.

The DSC algorithm keeps the best solution from each iteration at the first position until it is replaced by a better one.

Note that the maximum average rate of iterations was especially high (561) for the Schwefel function (2-D) for 92% success rate, for which the classical genetic algorithm failed to find solution, see Table 3.4.

Table 3.9 shows the test of DSC algorithm on 10-dimensional problems (Sum Squares, Sphere function, Sum of Different Powers, Zakharov, Rastrigin) with 160 chromosomes and the number of iterations fixed to 2000.

It should be noted that, in addition to the experiments reported here, it is proved in research [91] that the DSC algorithm is superior over the CGA for the problem of minimizing a global scalarization function of a multiobjective optimization problem (a global scalarization function is introduced in [92]).

Figures 3.3-3.6 present the average number of iterations with standard deviation of iterations for 2-dimensional functions by using 40, 80 and 160 chromosomes for DSC algorithm, also for 10-dimensional functions. Section 3.7 contains the processing time of DSC algorithm on tested function.

Finally, the execution time of the DSC algorithm displayed as output. A computer with 2.4 MHz core i5, 8 GB RAM was used. In Table 3.3-3.5 and Table 3.9 we show the minimum, maximum and average run time in seconds for all tested functions.

Table 3. 3 The results for 50 runs of the DSC algorithm (40 chromosomes).

Function name	Threshold of stopping criteria	Min number of iteration / Min time in seconds	Max number of iteration / Max time in seconds	Mean no. of iterations for all successful runs / Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Rate of success DSC	Rate of success GA
Easom	0.001	31	464	181	95.6	-0.99543	100%	100% DV
		0.0139	0.1450	0.0673				
Matyas	0.001	4	391	64	67.3	0.000505	100%	100% DV
		0.0046	0.1214	0.0214				

Beale's	0.001	5	1349	179	230.1	0.000517	100%	70% DV
		0.0053	0.4730	0.0663				
Booth's	0.005	13	1181	321	338.6	0.00247	90%	100% DV
		0.0051	0.4091	0.1682				
Goldstein –Price	0.001	46	896	287	189.1	3.00038	100%	100% DV
		0.0197	0.3907	0.1282				
Schaffer N.2	0.001	24	1533	476	360.8	4.11E-05	100%	70% DV
		0.0163	0.414345	0.137				
Schwefel's	0.01	94	2390	506	574.1	0.07317	50%	0% BS
		0.0367	0.7968	0.3176				
Branins's rcos	0.001	16	2332	171	406.3	0.39853	100%	100% DV
		0.0152	0.5922	0.069				
Six-hump camel back	0.001	9	215	73	54.3	-1.03125	100%	100% DV
		0.0100	0.0644	0.0295				
Shubert	0.01	5	149	67	60.4	-186.715	100%	100% DV
		0.0043	0.0977	0.0200				
Martin and Gaddy	0.001	7	438	53	64.4	3.95E-05	100%	40% DV
		0.0057	0.0959	0.0191				
Michalewicz	0.04	40	1500	346	319.3	38.81764	100%	80% DV
		0.0166	0.3666	0.0975				
Holder table	0.001	9	535	100	95.6	-19.2035	100%	80% DV
		0.0086	0.0775	0.0336				
Drop-wave	0.001	30	1621	420	432.2	-0.99517	100%	100% BS
		0.0134	0.5932	0.1520				
Levy N. 13	0.001	47	1700	504	437.5	0.000583	98%	100% BS
		0.0161	0.714964	0.159699				
Rastrigin's	0.001	16	330	127	79.2	0.00505	100%	100% BS
		0.0087	0.0982	0.0366				
sphere	0.001	15	395	155	98.5	0.003588	100%	100%

		0.0129	0.0927	0.0435				BS
Rosenbrock valley	0.001	5	896	270	167	0.000564	100%	100% BS
		0.0104	0.1971	0.0650				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

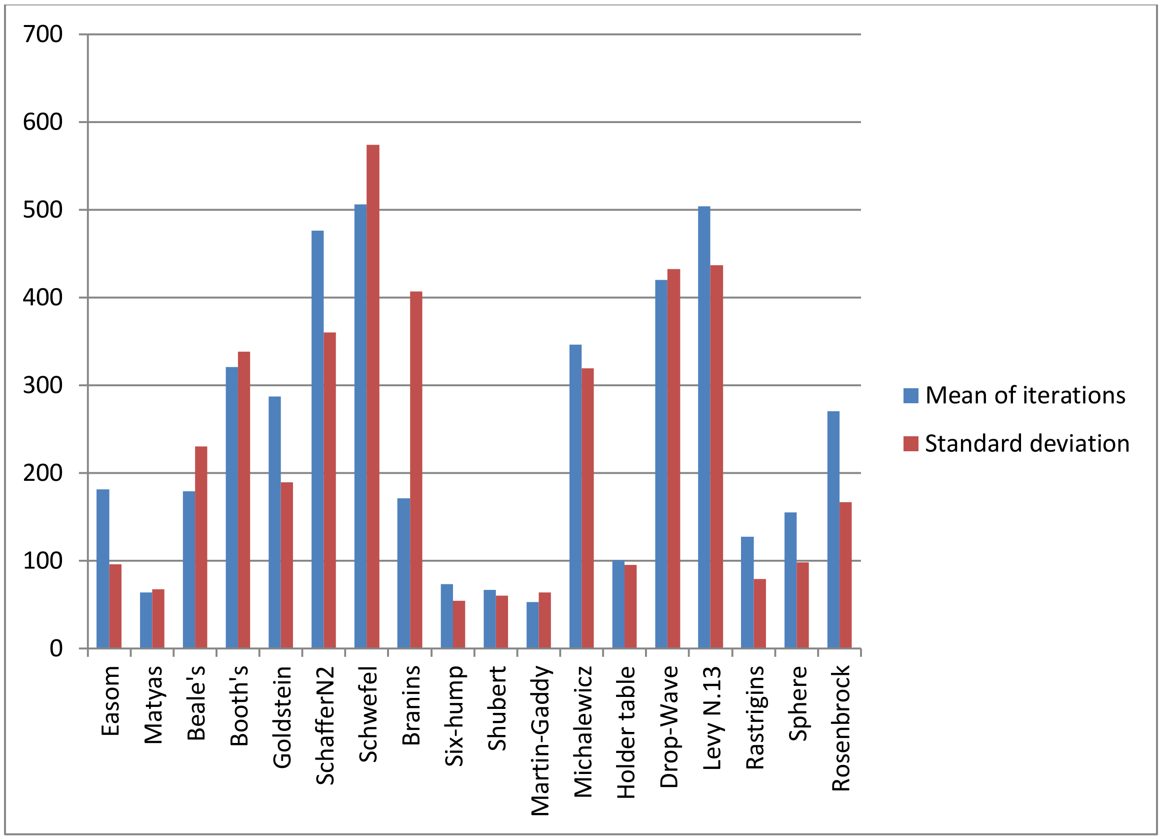


Figure 3. 3 The average number of iterations and standard deviation of iterations for 2-dimensional functions with 40 chromosomes for DSC algorithm

Table 3. 4 The results for 50 runs of the DSC algorithm (80 chromosomes).

Function name	Threshold of stopping criteria	Min number of iterations / Min time in seconds	Max number of iterations / Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter	Mean of the best solution fitness from all successful runs	Rate of success DSC	Rate of success GA
Easom	0.001	16	286	88	49.2	-0.99579	100%	100% DV
		0.0113	0.1500	0.0570				
Matyas	0.001	6	72	31	20.6	0.000492	100%	100% DV
		0.0029	0.0322	0.0228				
Beale's	0.001	4	646	93	105	0.00059	100%	70% DV
		0.0064	0.3881	0.0592				
Booth's	0.001	5	980	151	202	0.003198	100%	100% DV
		0.0079	0.6063	0.1041				
Goldstein–Price	0.001	11	242	134	78	3.00036	100%	100% DV
		0.0085	0.1203	0.0481				
Schaffer N.2	0.001	5	605	278	244	4.11E-05	100%	70% DV
		0.0055	0.2685	0.0942				
Schwefel's	0.01	51	1829	561	599	0.015643	92%	0% BS
		0.0273	1.1530	0.5606				
Branins's rcos	0.001	3	580	86	120	0.39853	100%	100% DV
		0.0063	0.4356	0.0451				
Six-hump camel back	0.001	2	115	39	26.4	-1.03129	100%	100% DV
		0.0036	0.0511	0.0183				
Shubert	0.01	11	773	198	224	-186.716	100%	100% DV
		0.0102	0.4209	0.1144				
Martin and Gaddy	0.001	6	151	36	34	3.02E-05	100%	40% DV
		0.0046	0.1002	0.0152				

Michalewicz	0.04	26	713	207	164	38.81257	100%	80% DV
		0.0156	0.4407	0.1228				
Holder table	0.001	4	163	47	36	-19.8125	100%	80% DV
		0.0052	0.1077	0.0310				
Drop-wave	0.001	13	816	201	176	-0.99487	100%	100% BS
		0.0130	0.3333	0.0902				
Levy N. 13	0.001	29	816	290	169	0.000547	100%	100% BS
		0.0167	0.3892	0.1958				
Rastrigin's	0.001	14	181	71	89.6	0.007197	100%	100% BS
		0.0156	0.1576	0.0429				
sphere	0.001	19	186	75	42.3	0.004133	100%	100% BS
		0.0163	0.1158	0.0376				
Rosenbrock's valley	0.001	5	438	101	93.3	0.00059	100%	100% BS
		0.0100	0.1532	0.0433				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

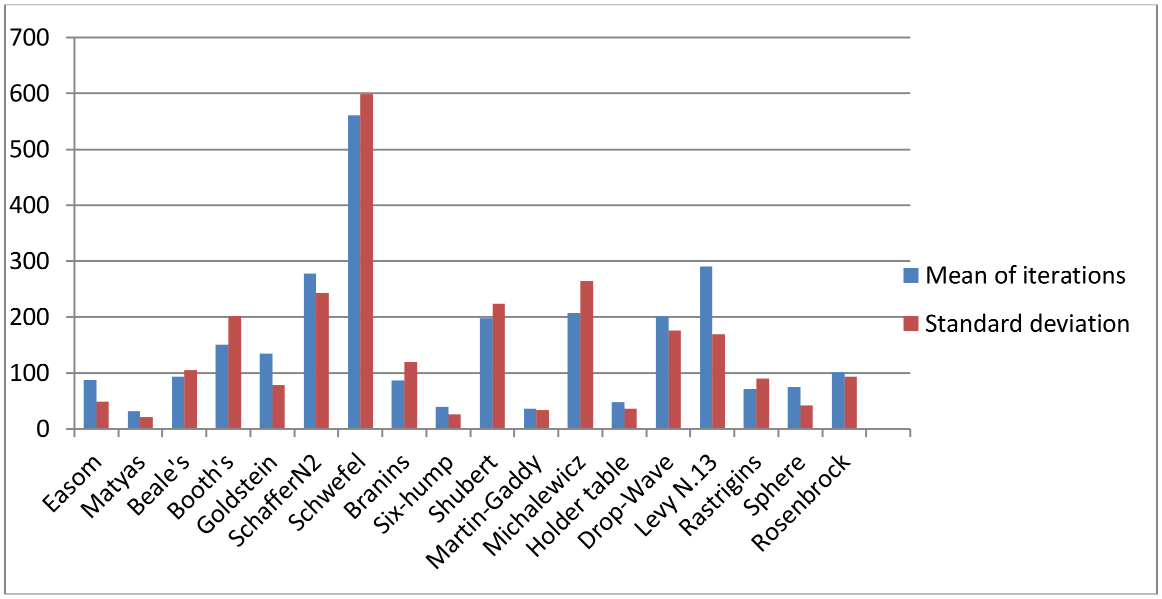


Figure 3. 4 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for DSC algorithm

Table 3. 5 The results for 50 runs of the DSC algorithm (160 chromosomes).

Function name	Threshold of stopping criteria	Min number of iteration/ Min time in seconds	Max number of iteration/ Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Rate of success DSC	Rate of success GA
Easom	0.001	11	141	61	28.6	-0.99927	100%	100% DV
		0.0165	0.1223	0.0508				
Matyas	0.001	2	29	13	7.7	0.000434	100%	100% DV
		0.0089	0.0266	0.0174				
Beale's	0.001	2	212	48	46.2	0.000523	100%	70% DV
		0.0087	0.1426	0.0410				
Booth's	0.001	6	1018	123	174.4	0.000595	100%	100% DV
		0.0124	0.6386	0.0856				
Goldstein–Price	0.001	12	106	44	22.5	3.000484	100%	100% DV
		0.0164	0.0725	0.0360				
Schaffer N.2	0.001	6	731	107	133.4	0.00045	100%	70% DV
		0.0127	0.5040	0.0808				
Schwefel's	0.01	26	2301	517	834.2	0.07051	100%	0% BS
		0.0275	1.8039	0.5735				
Branins's rcos	0.001	2	324	40	59.4	0.398517	100%	100% DV
		0.0089	0.2122	0.0347				
Six-hump camel back	0.001	1	41	14	9.7	-1.03106	100%	100% DV
		0.0049	0.0352	0.0178				
Shubert	0.01	10	457	111	104.3	-186.716	100%	100% DV
		0.0170	0.3327	0.0847				
Martin and Gaddy	0.001	3	38	14	9	0.000513	100%	40% DV
		0.0044	0.0325	0.0178				

Michalewicz	0.04	6	297	93	69.2	38.81715	100%	80% DV
		0.0128	0.2089	0.0683				
Holder table	0.001	6	725	84	113	-19.2078	100%	80% DV
		0.0133	0.4730	0.0624				
Drop-wave	0.001	14	708	122	144.2	-0.99954	100%	100% BS
		0.0178	0.4518	0.0857				
Levy N. 13	0.001	4	538	117	113.5	0.000471	100%	100% BS
		0.0112	0.3565	0.0840				
Rastrigin's	0.001	17	116	53	22.3	0.000442	100%	100% BS
		0.0201	0.0810	0.0429				
sphere	0.001	1	42	12	17.2	0.000445	100%	100% BS
		0.0018	0.0347	0.0166				
Rosenbrock's valley	0.001	5	199	45	53.3	0.000533	100%	100% BS
		0.0119	0.1279	0.0364				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

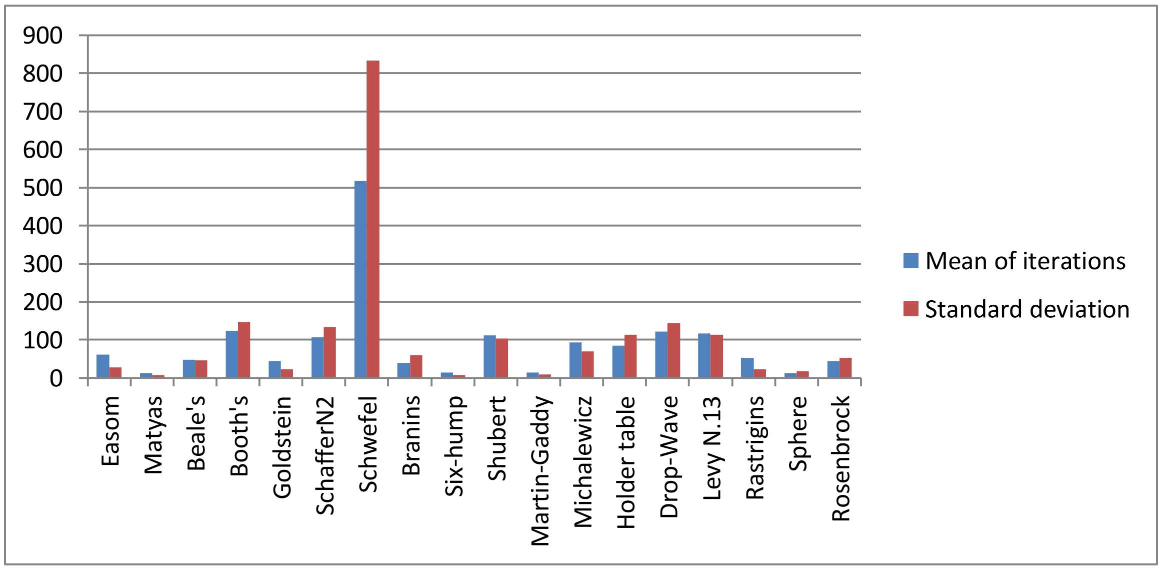


Figure 3. 5 The average number and standard deviation of iterations for 2-dimensional functions with 160 chromosomes for DSC algorithm

Table 3. 6 Comparing the mean number of iterations and success rate of functions for 50 runs of the algorithm (40 vs 80 vs 160 chromosomes).

Function name	Mean no. of iterations for all successful runs 40 ch.	Mean no. of iterations for all successful runs 80 ch.	Mean no. of iterations for all successful runs 160 ch.	Rate of success DSC (40 ch.)	Rate of success DSC (80 ch.)	Rate of success DSC (160 ch.)	Rate of success GA
Easom	349	88	61	100%	100%	100%	100% DV
Matyas	40	31	13	100%	100%	100%	100% DV
Beale's	217	93	48	98%	100%	100%	70% DV
Booth's	528	151	123	98%	100%	100%	100% DV
Goldstein–Price	182	134	44	100%	100%	100%	100% DV
Schaffer N.2	239	278	107	100%	100%	100%	70% DV
Schwefel's	1554	561	557	60%	92%	100%	0% BS
Branins's rcos	152	86	40	100%	100%	100%	100% DV
Six-hump camel back	58	39	14	100%	100%	100%	100% DV
Shubert	500	198	111	100%	100%	100%	100% DV
Martin and Gaddy	53	36	14	100%	100%	100%	40% DV
Michalewicz	395	207	93	100%	100%	100%	80% DV
Holder table	304	47	84	100%	100%	100%	80% DV
Drop-wave	505	194	122	100%	100%	100%	100% BS
Levy N. 13	452	209	117	100%	100%	100%	100% BS
Rastrigin’s	127	71	53	100%	100%	100%	100% BS

sphere	155	75	12	100%	100%	100%	100% BS
Rosenbrock's valley	270	101	45	100%	100%	100%	100% BS

BS= bit string, DV= double vector as a parameter of population type in GA toolbox.

Table 3.7 presents a comparative study of success rate and the number of function evaluations for the CMA-ES (Covariance Matrix Adaptation Evolution Strategy), DE (Differential Evolution) and DSC algorithms; it shows the DSC algorithm is the most successful one (see, especially, the Drop-wave function). The Matlab codes for the CMA-ES and DE algorithms were taken from [93] and [94], respectively. We have used 80 chromosomes 2500 iterations for all.

Table 3. 7 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and DSC algorithms (50 runs, max 2500 iterations, 80 chromosomes).

function name	**CMA-ES success rate**	**Function evaluations of CMA-ES**	**DE success rate**	**Function evaluations of DE**	**DSC success rate**	**Function evaluations of DSC**
Easom	70%	17053	100%	3240	100%	7588
Matyas	100%	500	100%	2700	100%	2480
Beale	100%	460	100%	3060	100%	7440
Booth's	100%	492	100%	2820	100%	12080
Goldstein–Price	100%	1812	100%	1620	100%	10720
Schaffer N.2	90%	6726	100%	5016	100%	8356
Schwefel's	0%	----	0%	----	**92%**	**44880**
Branins's rcos	100%	6876	100%	840	100%	6880
Six-hump camel	100%	780	100%	2160	100%	3120
Shubert	90%	2220	100%	8160	100%	15840

Martin and Gaddy	100%	1660	100%	2400	100%	2880
Michalewicz	100%	1848	0%	---	100%	16560
Drop-wave	50%	26470	94%	9048	100%	13788
Levy N. 13	100%	606	100%	1958	100%	9216
Rastrigin's	80%	13134	100%	2388	100%	8022
Sphere	100%	720	100%	1800	100%	4500
Ackley d=4	100%	2240	100%	3480	100%	30240
Rosenbrock's	100%	1644	100%	4560	100%	8080
Sum Squares d=10	100%	3600	100%	6200	25%	309760
Sphere d=10	100%	3840	100%	9200	100%	119360
Sum of Different Powers d=10	100%	480	100%	4300	100%	2240
Zakharov d=10	0%	---	100%	124400	12%	289280
Rastrigin d=10	0%	---	100%	7200	0%	----

Table 3.8 presents the number of bits that were used for each function depending on the size of range for (x_1, x_2). This number was calculated by using the difference of upper and lower bound of the domain multiplied by 10000 to divide the domain to small parts, i.e., $(b_i - a_i) * 10000$ for each x_i . Then, to find the appropriate number of bits, we find the smallest integer m_i such that $(b_i - a_i) * 10000 \leq \ 2^{m_i} - 1$ (see p. 33 of the Michalewicz book [87]). For example for the Easom function we have, for both $x_1$and x_2 (100-(-100)) *10000 =2000000 and $2^{21} - 1 = 2097152$, so this range is represented by 21 bits.

Table 3. 8 The number of bits for each function.

Function name	No. of bits for x_1	No. of bits for x_2
Easom	21	21
Matyas	18	18
Beale's	17	17
Booth's	18	18
Goldstein–Price	16	16
Schaffer N.2	21	21
Schwefel's	24	24
Branins's rcos	18	18
Six-hump camel back	16	16
Shubert	18	18
Martin and Gaddy	17	17
Michalewicz	18	15
Holder table	18	18
Drop-wave	17	17
Levy N. 13	18	18
Rastrigin's	17	17
sphere	17	17
Ackley d=4	20	20
Sum Squares	18	18
Sum of Different Powers	15	15
Zakharov	18	18
Rosenbrock's valley	16	16

Table 3.9 presents the best value of 10-dimensional functions for 25 runs of the DSC algorithm, here we used 160 chromosomes and the number of iterations was fixed to 2000, with execution time shown under the respective number of iterations.Figure 3.6 represents the average number of iterations with standard deviation of iterations for 10-dimensional functions by using 160 chromosomes for DSC algorithm.

Table 3. 9 The results for 25 runs of the DSC algorithm for 10-dimensional functions (160 chromosomes) with execution time.

Function name	Threshold	Min number of iteration/ Min time in seconds	Max number of iteration / Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Rate of success DSC	Rate of success GA
Sum Squares d=10	0.1	1421	1989	1936	151.9	0.21577	25%	100% BS
		2.0701	3.3096	2.8726				
Sphere d=10	0.1	359	1396	746	272.9	0.09027	100%	100% BS
		0.5064	1.954	1.0558				
Sum of Different Powers d=10	0.1	2	39	14	11.3	0.02001	100%	100% BS
		0.003426	0.08239	0.0243				
Zakharov d=10	0.1	449	1992	1808	539.3	0.63340	12%	100% BS
		0.9185	3.9840	2.8476				
Rastrigin d=10	0.1	2000	2000	2000	0	15.32815	0%	100% BS
		2.6243	2.9219	2.6687				
Ackley d=4	0.001	123	939	378	203	0.071314	100%	100% BS
		0.1589	1.2559	0.5032				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

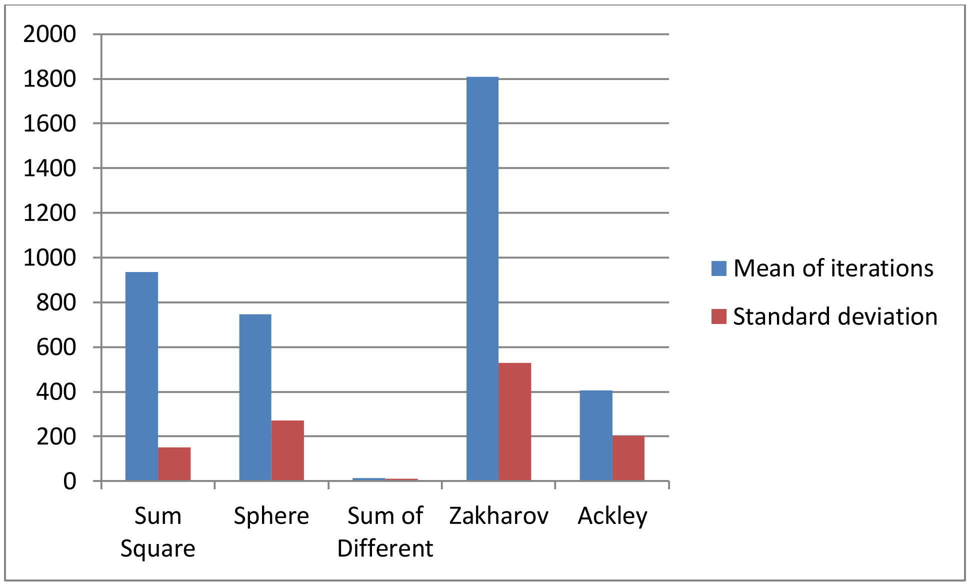


Figure 3. 6 The average number and standard deviation of iterations for 10-dimensional functions with 160 chromosomes for DSC algorithm

3.6 Discussion of figures

Figure 3.7 shows a two-dimensional view of the Easom function. It can be seen that the DSC algorithm has reached the best solution at the blue point at $f(\pi, \pi) = -1$. Figure 3. 8 shows a two-dimensional view of Schaffer's function. It can be seen that DSC algorithm has reached the best solution at the blue point on the focus view in the right upper corner of the figure. For this function, it is difficult to reach an optimal solution because it contains multiple local minima near to the best one.

Figures 3.9-3.16 show two-dimensional views of Shubert problem with 18 optimal solution points, Branins's problem with 3 optimal solutions, Six-hump camel back problem with two optimum points and Holder table problem with 4 optimum points. For the remaining problems: Michalewicz problem, Drop-wave problem, Schwefel's problem, Levy N.13 problem there is only one optimal solution for each.

Figures 3.17, 3.18 show how the best fitness values of the population evolve with the number of iterations. Here the red colour means jumping to a better solution.

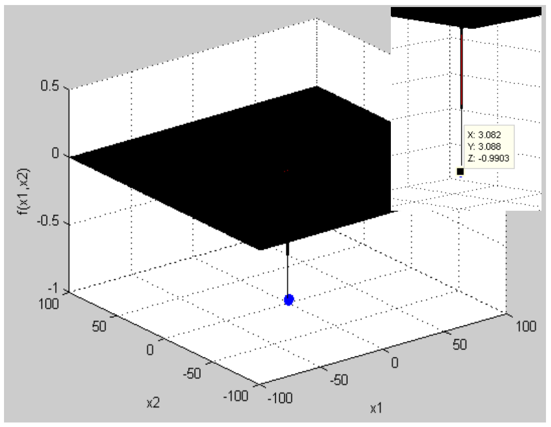


Figure 3. 7 Solutions of Easom Pr..

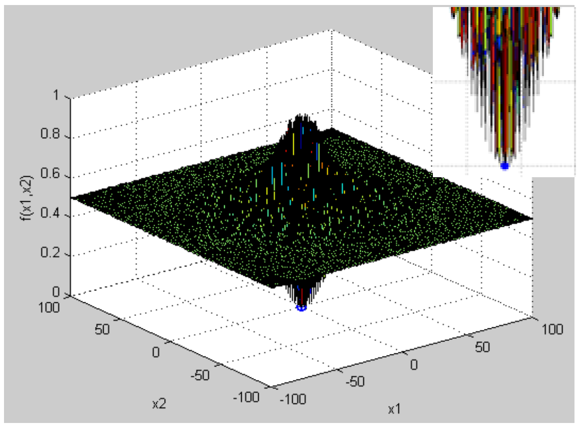


Figure 3. 8 Solutions of Schaffer's problem.

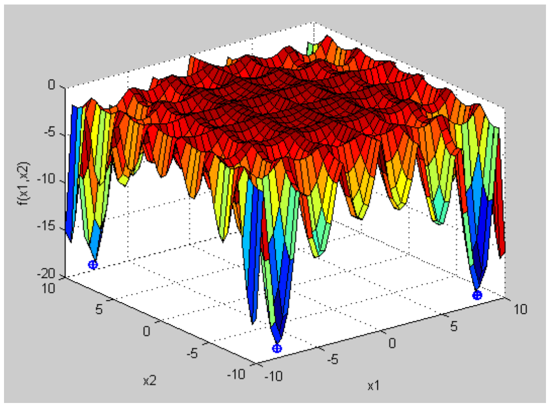


Figure 3. 9 Solutions of Holder-table.

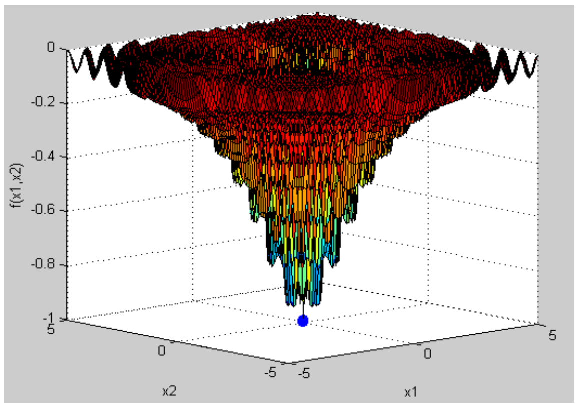


Figure 3. 10 Solutions of Drop-wave problem.

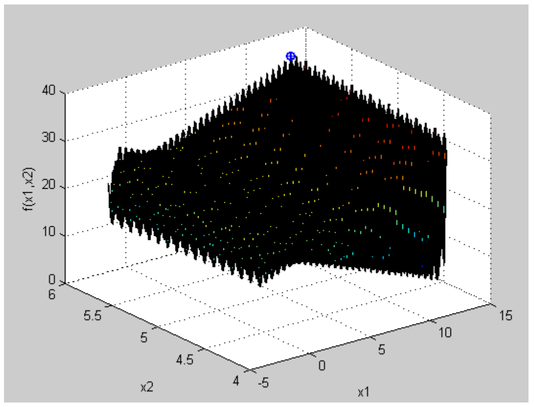


Figure 3. 11 Solutions of Michalewicz pr.

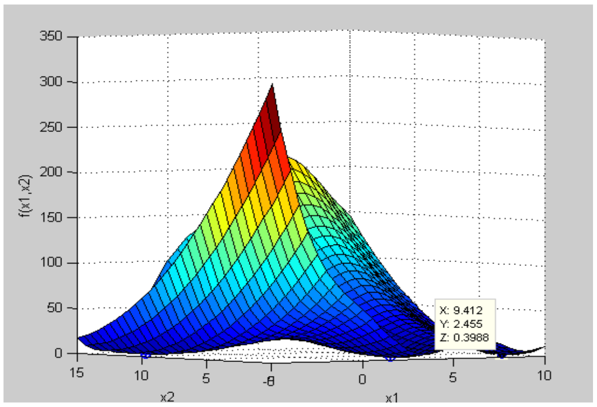


Figure 3. 12 Solutions of Branins's problem.

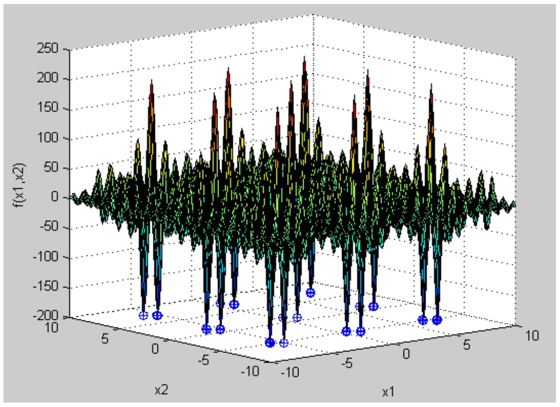


Figure 3. 13: Solutions of Shubert problem: 18 optimal solutions.

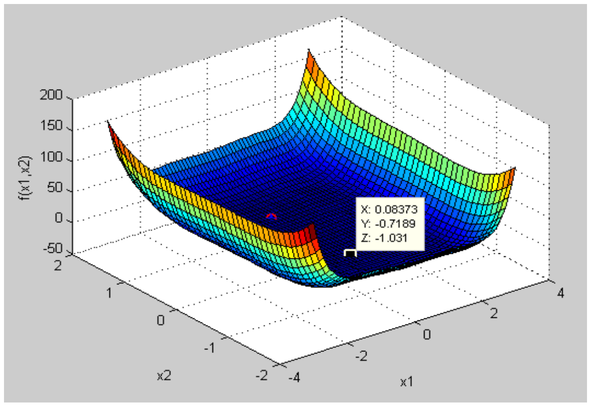


Figure 3. 14 Solutions of Six-hump camel back problem.

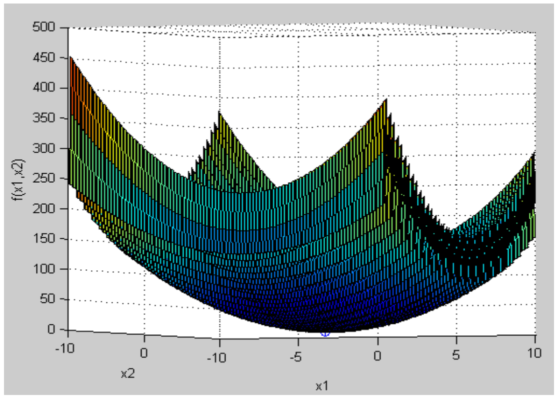


Figure 3. 15 Solutions of Levy N.13 pr.

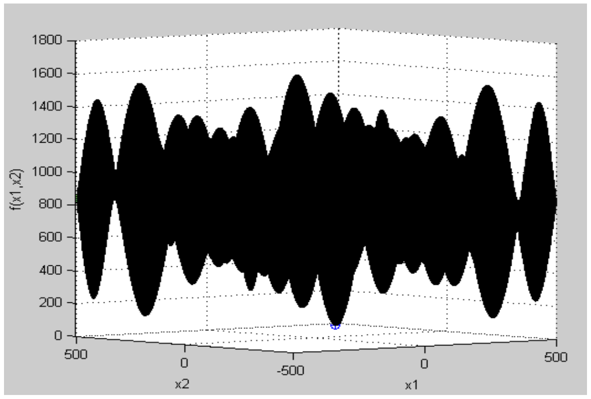


Figure 3. 16 Solutions of Schwefel's Pr.

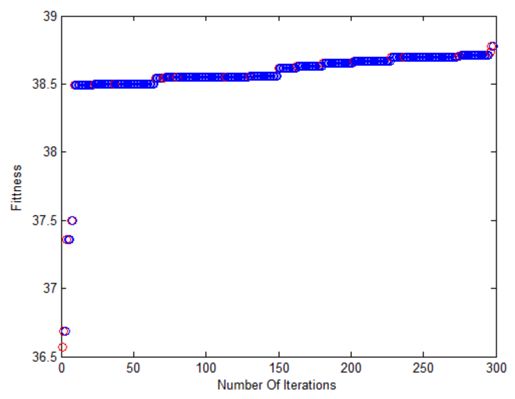


Figure 3. 17 Finding the best solution for Michalewicz problem in 300 iterations.

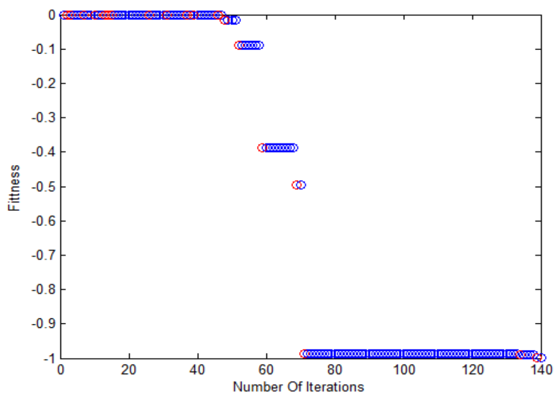
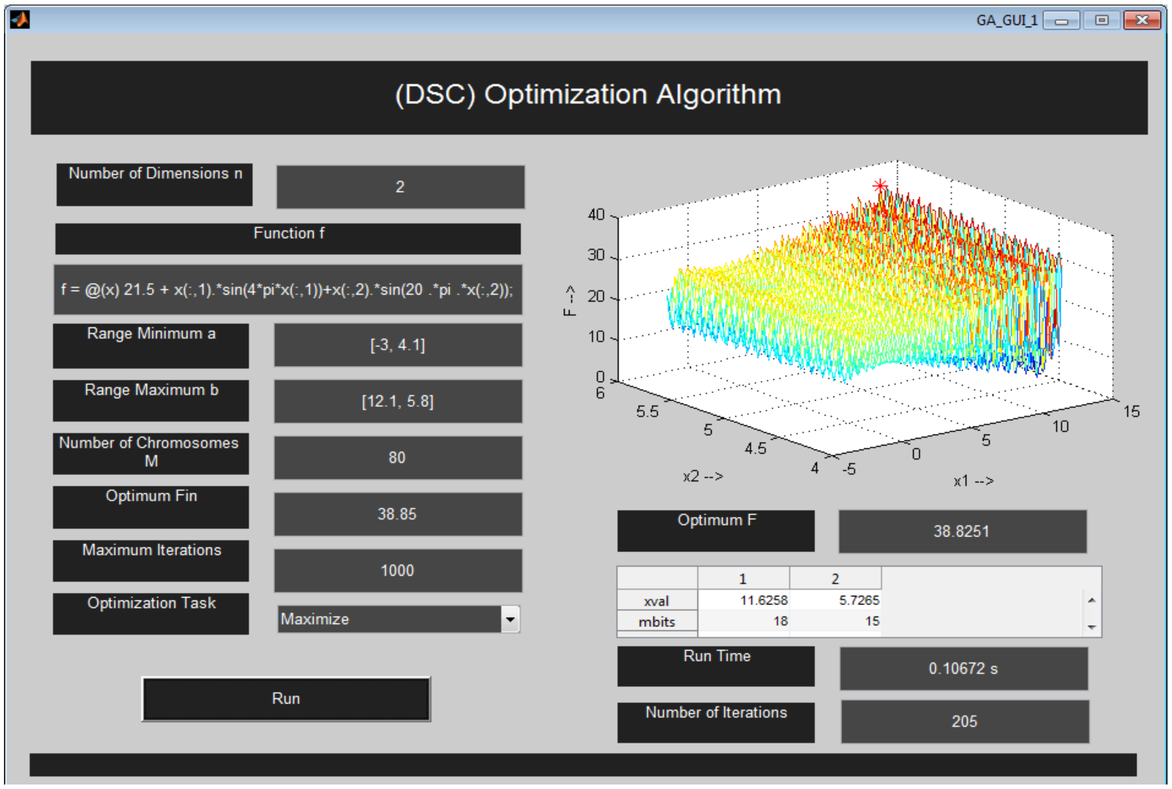


Figure 3. 18 Finding the best solution for Schaffer problem in 140 iterations.

Figure 3.19 shows the Graphical User Interface (GUI) for DSC algorithm, that was designed and programed by the author. It contains inputs for function description, number of dimensions, range of $[a_i, b_i]$ for the function under test, number of elements (chromosomes), maximum number of iterations, a choice box for minimum or maximum. Also, the results will output at the right side as follows: the graph of a function, the best value for $f(x)$ with values of x_i, the number of bits that are used to represent a solution, execution time and the number of iterations. This figure shows the Michalewicz problem.



Number of Dimensions n: 2

Function f: f = @(x) 21.5 + x(:,1).*sin(4*pi*x(:,1))+x(:,2).*sin(20 .*pi .*x(:,2));

Range Minimum a: [-3, 4.1]

Range Maximum b: [12.1, 5.8]

Number of Chromosomes M: 80

Optimum Fin: 38.85

Maximum Iterations: 1000

Optimization Task: Maximize

Optimum F: 38.8251

	1	2
xval	11.6258	5.7265
mbits	18	15

Run Time: 0.10672 s

Number of Iterations: 205

Figure 3. 19 shows the Graphical User Interface (GUI) for the DSC algorithm for the Michalewicz function.

3.7 Experimental results of GA

In the GA toolbox we used the following options (they also apply to later chapters):

1. Population type specifies the type of the input to the fitness function. We used bit string \ double vector .
2. Population size = 80 chromosomes
3. Creation function = feasible population. “GA creates a random initial population using a creation function. We can specify the range of the vectors in the initial population in the Initial range field in Population options. Feasible population creates a random initial population that satisfies all bounds and linear constraints” [95].
4. Initial population = default. “The algorithm begins by creating a random initial population, the default value of Initial range in the Population options is [0;1]” [95].
5. Fitness scaling = Rank
6. Selection = Roulette
7. Mutation function = Uniform.— Uniform mutation is a two-step process. First, the algorithm selects a fraction of the vector entries of an individual for mutation, where each entry has a probability Rate of being mutated. “The default value of Rate is 0.01. In the second step, the algorithm replaces each selected entry by a random number selected uniformly from the range for that entry” [95].
8. Crossover function = two point.
9. Stopping criteria = 2500 iterations.
10. Fitness limit = 0.001 is the threshold of stopping criteria (for most functions).

Table 3.10 presents the experimental results of GA success rate and mean number of iterations by using 80 chromosomes and maximum 2500 iterations, on two types of population (bit string, Double vector). These results used in our comparison with all algorithms in addition to CMA-ES and DE algorithms.

Table 3. 10 Comparing the success rate and the mean number of iterations for the GA, first with Bit string and next with Double vector parameter

Function name	Threshold	Rate of success GA Bit string	Mean no. of iterations for all successful runs with bit string	Rate of success GA Double vector	Mean no. of iterations for all successful runs with double vector
Easom	0.001	0%	---	100%	124
Matyas	0.001	90%	220	100%	125
Beale's	0.001	0%	---	70%	204
Booth's	0.001	0%	---	100%	75
Goldstein–Price	0.001	0%	---	100%	82
Schaffer N.2	0.001	0%	---	70%	93
Schwefel's	0.001	0%	---	0%	---
Branins's rcos	0.001	0%	---	100%	68
Six-hump camel back	0.001	0%	---	100%	75
Shubert	0.01	0%	---	100%	64
Martin and Gaddy	0.001	0%	---	40%	320
Michalewicz	0.04	20%	95	80%	72
Holder table	0.001	0%	---	80%	240
Drop-wave	0.001	100%	51	0%	---
Levy N. 13	0.001	100%	51	0%	---
Rastrigin’s	0.001	100%	51	100%	51
Sphere	0.001	100%	51	50%	63
Ackley d=4	0.001	100%	51	0%	---
Rosenbrock’s valley	0.001	100%	51	0%	---

3.8 Convergence of DSC

In this section we present a theorem on the convergence in mean of the DSC algorithm. The idea of the proof is similar to that of [64] but a more rigorous mathematical formulation is given here, especially concerning the considered probability space. In our considerations, we will use the theory of denumerable stochastic processes described in [96]. Below we consider the minimization problem only.

Let n be the size of population P, and m – the number of bits in a chromosome. Let Z be the set of all chromosomes of size m. Then the search space of the DSC algorithm is the following finite set:

$$S = \underbrace{Z \times Z \times \ldots \times Z}_{n \text{ members}}$$

S is the set of all possible populations of size n, where each population is a sequence of n bit strings of the same length m. Each population can also be considered as one bit string concatenated from all chromosomes in the population.

Denote by P^k the population of the DSC algorithm after k iterations ($k = 1,2,\ldots$). Let $f\colon \{0,1\}^m \to \mathbb{R}$ be the fitness function of the algorithm. Define a function $\bar{f}$ on populations P^k by

$$\bar{f}(P^k) \coloneqq \min_{s \in P^k} f(s)$$

In the sequel, the mean of a random variable g will be defined by $M[g] \coloneqq \int_{\Omega} g \, d\mu$ where μ is the measure associated with some probability space. To be able to compute the mean of the $\bar{f}$, we must show that it is a measurable simple function. Of course, $\bar{f}$ has only a finite number of values because there is only a finite number of possible populations.

We now define a denumerable stochastic process for the DSC algorithm. Let Ω be a sequence space ([96], p. 43) whose elements are of the form $\omega = (\omega_0, \omega_1, \ldots)$, where $\omega_0, \omega_1, \ldots$ are elements of S, ω_0 is the initial population, and ω_k is the population obtained in iteration k of the algorithm:

$$\Omega = \{\omega = (\omega_0, \omega_1, \ldots) \mid \forall_i \; \omega_i \in S\}$$

We define the k-th outcome function as follows:

$$x_k(\omega) = x_k(\omega_0, \omega_1, \ldots) := \omega_k$$

Let $\mathcal{F}_k$ be the family of all unions of subsets of Ω of the form

$$B_k := \{\omega | x_0(\omega) \in S_0 \wedge \ldots \wedge x_k(\omega) \in S_k\} \tag{1}$$

where $S_0, \ldots, S_n$ are some subsets of S. Observe that

$$\Omega \backslash B_k = \{\omega | x_0(\omega) \in S \backslash S_0 \vee \ldots \vee x_k(\omega) \in S \backslash S_k\} = \bigcup_{i=0}^{k} \{\omega | x_i(\omega) \in S \backslash S_i\} \in \mathcal{F}_k$$

so it is easy to verify that $\mathcal{F}_k$ is a Borel field. We will prove that $\mathcal{F}_0 \subset \mathcal{F}_1 \subset \mathcal{F}_2 \subset \cdots$ Take any set B_k of the form (1). We have

$$B_k = \{\omega | x_0(\omega) \in S_0 \wedge \ldots \wedge x_k(\omega) \in S_k \wedge x_{k+1}(\omega) \in S\}$$

Define

$$A_k := \{\omega | x_0(\omega) \in S_0 \wedge \ldots \wedge x_k(\omega) \in S_k \wedge x_{k+1}(\omega) \in S_{k+1}\}$$

where $S_{k+1} \subset S$ is an arbitrary set. Then, of course, the Borel field $\mathcal{F}_k$ generated by all sets of the form B_k is included in the Borel field $\mathcal{F}_{k+1}$ generated by all sets of the form A_k.

To construct a denumerable stochastic process on Ω, we must define a sequence of functions $\{f_k\}$ such that, for every fixed *k* and for each $s \in S$, the set $\{\omega | f_k = s\}$ is a set in $\mathcal{F}_k$. We can achieve this by taking $f_k(\omega) := x_n(\omega)$ (the *k*-th outcome function; see [96], p. 47). Now define

$$\mathcal{F} := \bigcup_{k=1}^{\infty} \mathcal{F}_k$$

It can be shown that $\mathcal{F}$ is not a Borel field. Let $\mathcal{G}$ be the smallest Borel field containing $\mathcal{F}$. Consider a basic cylinder set

$$B^{(k)} := \{\omega \in \Omega \, | x_0(\omega) = s_0 \wedge \ldots \wedge x_k(\omega) = s_k\}$$

where $s_0, \ldots, s_k \in S$. We define a measure ν on basic cylinder sets as follows:

$$\nu(B^{(k)}) := \Pr(c_1 | c_0) \cdot \Pr(c_2 | c_1) \cdot \ldots \cdot \Pr(c_k | c_{k-1})$$

where $\Pr(c_i | c_{i-1})$ is the probability that the DSC algorithm will generate population c_i in iteration i under the condition that it has generated population c_{i-1} in iteration $i-1$. It can be shown ([96], p.43) that ν can be uniquely extended to the sets of $\mathcal{F}$. It is also known that ν can be extended to a measure μ on the smallest Borel field $\mathcal{G}$ containing $\mathcal{F}$ ([96], p.43). Now we define $\mathcal{B}$ by adding to $\mathcal{G}$ all subsets of sets of measure μ zero. Then we extend μ to be a measure on $\mathcal{B}$ as follows: let $A \in \mathcal{B}$, then $A = B \cup C$ where $B \in \mathcal{G}$ and $C \subset D$for some $D \in \mathcal{G}$, where $\mu(D) = 0$. Then we define $\mu(A) \coloneqq \mu(B)$.

We have thus constructed a probability space $(\Omega, \mathcal{B}, \mu)$. From now on, we will use the notation Pr instead of μ. Observe that the population of the DSC algorithm constructed in iteration k depending on event ω is given by

$$P^k(\omega) = x_k(\omega) = \omega_k$$

We now define a random variable $\lambda: \Omega \to \mathbb{R}$ as follows:

$$\lambda(\omega) \coloneqq \bar{f}(P^k(\omega)) = \min_{s \in P^k(\omega)} f(s)$$

The mean of λ can be computed by

$$M\bar{f}(P^k) = \textstyle\sum_{\lambda \in \mathbb{R}} \lambda \cdot p_\lambda^k \qquad (2)$$

where

$$p_\lambda^k \coloneqq \Pr\left\{\omega \in \Omega |\, \bar{f}\left(P^k(\omega)\right) = \lambda\right\}$$

Observe that the sum in (2) has only a finite number of nonzero terms. Moreover, we have

$$M\bar{f}(P^k) = \sum_{\lambda \in D^k} \lambda \cdot \Pr\{\omega \in \Omega \mid \bar{f}\left(P^k(\omega)\right) = \lambda\}$$

where $D^k = \{\lambda_1^k, \lambda_2^k, \dots \lambda_q^k\} \coloneqq \left\{\bar{f}\left(P^k(\omega)\right) \middle| \omega \in \Omega\right\}, \lambda_i \neq \lambda_j$for $i \neq j$.

Then the set Ω can be represented as

$$\Omega = \Omega_1^k \cup \Omega_2^k \cup \dots \cup \Omega_q^k \text{ and } \Omega_{i_1}^k \cap \Omega_{i_2}^k = \emptyset \text{ for } i_1 \neq i_2$$

where $\Omega_j^k \coloneqq \{\omega \in \Omega | \bar{f}\left(P^k(\omega)\right) = \lambda_j^k\}, j = 1, \dots, q$.

Then

$$M\bar{f}(P^k) = \sum_{j=1}^{q} \lambda_j^k \cdot \Pr(\Omega_j^k)$$

<u>Theorem</u> (convergence of DSC algorithm)

We have

$$M\bar{f}(P^k) \to f^* := \min_{z\in Z} f(z)$$

For the proof of the theorem, we will need the following

<u>Lemma</u>

Let $s^* \in Z$, and let P^j be a population generated in iteration j of the DSC algorithm. Then

$$\Pr\{s^* \notin \text{Rand}(P^j)\} = (1-\frac{1}{2^m})^{\frac{n}{2}}$$

where Rand(P^j) is the second half of P^j which is generated randomly.

Proof. Let B_i^j be the event that we do not generate s^* at the i-th random generation in iteration j $\left(i=1,\dots,\frac{n}{2}; j=1,2,\dots\right)$. Then $\Pr\left(\Omega\backslash B_i^j\right) = \frac{1}{2^m}$ (we generate s^* at a single random generation of a chromosome if and only if each bit of a generated chromosome is equal to the corresponding bit of s^*, which holds with probability ½ for each of m positions). This implies that $\Pr\left(B_i^j\right) = 1-\frac{1}{2^m}$. Then

$$\Pr\{s^* \notin \text{Rand}(P^j)\} = \Pr\left\{\bigcap_{i=1}^{\frac{n}{2}} B_i^j\right\} = \prod_{i=1}^{\frac{n}{2}} \Pr\{B_i^j\} = \prod_{i=1}^{\frac{n}{2}}\left(1-\frac{1}{2^m}\right) = \left(1-\frac{1}{2^m}\right)^{\frac{n}{2}} \blacksquare$$

Proof of the theorem. Define $\alpha := (1-\frac{1}{2^m})^{\frac{n}{2}}$. Without restriction of generality, we may assume that $f(z) \le 0$ for all $z \in Z$ (if this is not the case, we can add a suitable negative constant to f to achieve this inequality). Denote

$$p(k,f^*) := \Pr\left\{\omega \in \Omega | \bar{f}\left(P^k(\omega)\right) = f^*\right\}$$

Suppose that there are l individuals $s_1^*,\dots,s_l^*$ with fitness f^* in the space Z. Consider the event A^1 that no solution is found in the first iteration. Since

$$A^1 \subset \{s_1^*,\dots,s_l^* \notin \text{Rand}(P^1)\} \subset \{s_1^* \notin \text{Rand}(P^1)\}$$

we have

$$\Pr(A^1) \leq \Pr\{s_1^*, \dots, s_l^* \notin \text{Rand}(P^1)\} \leq \Pr\{s_1^* \notin \text{Rand}(P^1)\} = \left(1 - \frac{1}{2^m}\right)^{\frac{n}{2}} = \alpha$$

where the last equality follows from the Lemma. We have thus proved that $\Pr(A^1) \leq \alpha$. Now we can prove by induction that $\Pr(A^k) \leq \alpha^k$, where A^k is the event that no solution is found in the k-th iteration. Suppose that:

$$\Pr(A^{k-1}) \leq \alpha^{k-1} \qquad (3)$$

We will show that $A^k \subset A^{k-1}$. Indeed, if a solution is found in iteration $k-1$, then in iteration k it is moved by sorting procedure to the top of population, and it is not destroyed; therefore, the solution is also found in iteration k. Using this inclusion, we find that

$$\Pr(A^k | A^{k-1}) = \frac{\Pr(A^k \cap A^{k-1})}{\Pr(A^{k-1})} = \frac{\Pr(A^k)}{\Pr(A^{k-1})} \qquad (4)$$

Since $A_k \subset \{s_1^* \notin \text{Rand}(P^k)\}$ and the events $\{s_1^* \notin \text{Rand}(P^k)\}$ and A_{k-1} are independent, we obtain

$$\Pr(A_k | A_{k-1}) \leq \Pr(s_1^* \notin \text{Rand}(P^k) | A_{k-1}) = \Pr\{s_1^* \notin \text{Rand}(P^k)\} = \alpha \qquad (5)$$

where the last equality follows from the Lemma. Using condition (4), then conditions (3) and (5), we get

$$\Pr(A^k) = \Pr(A^k | A^{k-1}) \Pr(A^{k-1}) \leq \alpha \cdot \alpha^{k-1} = \alpha^k$$

We have thus proved by induction that

$$\Pr(A^k) \leq \alpha^k$$

Hence the probability that the solution has been found in iteration k can be estimated as follows:

$$p(k, f^*) = 1 - \Pr(A^k) \geq 1 - \alpha^k$$

Observe that, for each k, and for each $\lambda_i^k \in D_k$, $i \in \{1, \dots q\}$, we have $f^* \leq \lambda_i^k$.

Hence,

$$M\bar{f}(P^k) = \sum_{j=1}^{q} \lambda_j^k \Pr(\Omega_j^k) \geq f^* \sum_{j=1}^{q} \Pr(\Omega_j^k) = f^* \qquad (6)$$

We will also prove that

$$M\bar{f}(P^k) = \sum_{j=1}^{q} \lambda_j^k \Pr(\Omega_j^k) \leq f^* p(k, f^*) \qquad (7)$$

This inequality follows because one of the values $\lambda_1^k, \dots, \lambda_q^k$ is equal to f^* (a solution can always be selected at any iteration k), and for this value λ_j^k, we have $\Pr\left(\Omega_j^k\right) = p(k, f^*)$. Therefore, the term $f^* p(k, f^*)$ is one of the (non-positive) terms in the sum $\sum_{j=1}^{q} \lambda_j^k \Pr(\Omega_j^k)$.

Using inequalities (6) and (7) we obtain

$$f^* \leq M\bar{f}(P^k) \leq f^* p(k, f^*) \leq f^*(1 - \alpha^k) \xrightarrow[k \to \infty]{} f^*$$

This proves that $M\bar{f}(P^k) \to f^*$. ■

3.9 Conclusion

A new meta-heuristic optimization algorithm called Dissimilarity and Similarity of Chromosomes (DSC) is introduced. DSC can be simply implemented, without too many parameters. It includes two genetic operators (the dissimilarity and similarity operators), population sorting and random generation of a part of population. The experiments have shown quick convergence and good global searching ability of the algorithm. The DSC algorithm is easy to understand and uses a simple classical representation of points in $\mathbb{R}^n$.

The DSC algorithm has only one parameter to be set by the user: the number M of chromosomes. Therefore it is easier to test than the classical GA where the user must try multiple runs to test different combinations of parameters. For all the examples, 80 chromosomes are enough to solve the problem. As Table 3. 6 shows, there is a significant difference in the rate of success between 40 chromosomes and 80 chromosomes.

Table 3. 7 shows comparison of CMA-ES, DE and DSC algorithms in terms of mean number of function evaluations and success rate. We see that the CMA-ES and DE algorithms have not found the solution for Schwefel's function, but DSC algorithm has found the solution in 92% of success rate. However, for 10-dimensional test functions CMA-ES and DE are better than DSC for some functions.

CHAPTER FOUR: The DSDSC Algorithm

4.1 Introduction

This chapter presents an optimization algorithm called Dynamic Schema with Dissimilarity and Similarity of Chromosomes (DSDSC) which is a modification of the DSC algorithm described in the previous chapter. To show the effectiveness of the algorithm, it is tested and compared with the GA, CMA-ES and DE algorithms, it is run on 18 two-dimensional, one four-dimensional and five ten-dimensional optimization problems taken from literature. It has been found that, in most cases, the method is better than the classical genetic algorithm.

In the DSDSC algorithm, we use the notion of schema in another way. It is required that the schema has fixed high significant bit(s) for each variable x_i, then we put *'s on some of the remaining bits by using the similarity operator. This type of schema is used to determine the area of the solution in search space.

The DSDSC is (like the DSC before) inspired by the schema theory and the mechanism of similarity and dissimilarity of chromosomes. This procedure depends on dividing each generation into four equal parts and then applying different genetic operators to each of them. The presented algorithm is designed to find optimal solutions to numerical optimization problems.

This chapter is organized as follows. In Section 4.2 the methodology of the DSDSC algorithm are introduced. Section 4.3 describes the DSDSC algorithm and shows its flowchart. Section 4.4 gives the schema analysis of the algorithm. Section 4.5 gives the analysis of experimental results. Finally, conclusions are presented in Section 4.6.

4.2 Methodology of DSDSC algorithm

The DSDSC algorithm starts with a population of M elements representing a number of solutions to the problem. This population is divided into four equal groups and some different operators to these groups are applied. This will be discussed in Section 4.3.

Briefly, the DSDSC creates new chromosomes by exploring dynamic dissimilarity, similarity, dynamic schema and random generation of new chromosomes.

Table 4.1 shows all M chromosomes ($Ch_1 .. Ch_M$) divided into 4 groups (G1, G2, G3, G4).

Table 4. 1 All M chromosomes ($Ch_1 .. Ch_M$). Groups of chromosomes.

Ch_1 Ch. … Ch. … $Ch_{M/4}$	G1: To the first group the dynamic dissimilarity operator is applied.
$Ch_{M/4+1}$ Ch. … Ch. … $Ch_{M/2}$	G2: To the second group the similarity operator is applied.
$Ch_{M/2+1}$ Ch. … Ch. … $Ch_{M/2+M/4}$	G3: To the third group the dynamic schema operator is applied.
$Ch_{M/2+M/4+1}$ Ch. … Ch. … Ch_M	G4: The fourth group is generated randomly.

4.3 The DSDSC algorithm

The following optimization problem is considered:

$$f: \mathbb{R}^n \longrightarrow \mathbb{R}$$

$$\text{minimize}|\text{maximize } f(x_1, \dots, x_n) \text{ subject to}$$

$$x_i \in [a_i, b_i], i = 1, \dots, n$$

where $f: \mathbb{R}^n \longrightarrow \mathbb{R}$ is a given function.

In the algorithm described below, the encoding of chromosomes is the same as in Chapter 3.

Let M be a positive integer divisible by 8. The DSDSC algorithm consists of the following steps:

1) Generate M chromosomes, each chromosome representing a point $(x_1, \dots, x_n)$.
2) Compute the values of the fitness function f for each chromosome in the population.
3) Sort the chromosomes according to the descending (for maximization) or ascending (for minimization) values of the fitness function. Then divide the population into four equal groups (G1, G2, G3, G4).
4) Copy C times the first chromosome and put it in C positions in the first half of the population randomly, replacing the original chromosomes, where $C = M/8$.
5) Apply the dynamic schema operator to the chromosomes Ch_1 and $Ch_{M/4}$ (that is, the chromosomes on the positions 1 and $M/4$, respectively). This operator works as follows (see Table 4.2):

(a) First, divide each chromosome onto n parts corresponding to variables $(x_1, \dots, x_n)$, the i-th part having length m_i. Next, for each variable x_i, generate a random integer R_i from the set {3,…, $m_i/2$}. Define the “gray” part of x_i as the first segment of length R_i of the string corresponding to x_i. Define the “white” part of x_i as the second segment of length $m_i - R_i$ of the same string.

(b) For the “white” parts of both chromosomes, if the two bits are not equal, put a star (*) in the schema, then copy this schema $M/4$ times and put it in the third part of population (G3) between positions $M/2+1$ and $M/2+M/4$, then put randomly 0 or 1 in the positions having *. The positions marked in gray are kept unchanged.

Note. The name "dynamic schema operator" is justified by the fact that the lengths of "gray" and "white" segments of chromosomes may vary from iteration to iteration.

Table 4. 2 The dynamic schema operator

Before change: an example for finding schema from the first chromosome and the chromosome on position M/4. Here shadow bits are not destroyed.

No. of Ch.	m_1						m_2			
	R_1		$m_1 - R_1$				R_2	$m_2 - R_2$		
Ch_1	1	1	0	0	1	0	1	0	1	0
$Ch_{M/4}$	0	1	1	0	0	1	0	0	0	1
Schema	1	1	*	0	*	*	1	0	*	*

After finding the schema: put it in M/2…M/2+M/4 positions

$Ch_{M/2+1}$	1	1	*	0	*	*	1	0	*	*
$Ch_{M/2+2}$	1	1	*	0	*	*	1	0	*	*
Ch. …	1	1	*	0	*	*	1	0	*	*
Ch. …	1	1	*	0	*	*	1	0	*	*
$Ch_{M/2+M/4}$	1	1	*	0	*	*	1	0	*	*

After change: put randomly 0 or 1 in (*) bits

$Ch_{M/2+1}$	1	1	1	0	1	0	1	0	0	1
$Ch_{M/2+2}$	1	1	1	0	0	0	1	0	1	1
Ch. …	1	1	0	0	1	0	1	0	1	0
Ch. …	1	1	0	0	0	1	1	0	0	0
$Ch_{M/2+M/4}$	1	1	1	0	1	1	1	0	1	1

Compare pairs of chromosomes for the first half (G1, G2) of the population by applying the dynamic dissimilarity and similarity operators (see Table 4.3 and Table 3.2). Check each two following chromosomes, i.e. the first and the second, the second and the third, and so on, by comparing the respective bits, as follows:

(a) For chromosomes in the first quarter (G1) of the population (from 1 to $M/4$), apply the dynamic dissimilarity operator, dividing each chromosome onto n parts corresponding to variables $(x_1, \dots, x_n)$, the i-th part having length m_i. Next, for each variable x_i, generate a random integer R_i from the set {3,…, $m_i/2$}. Define the "gray" part of x_i as the first segment of length R_i of the string corresponding to x_i. Define the "white" part of x_i as the second segment of length $m_i - R_i$ of the same string. The "gray" part of x_i is not destroyed. In the white" part of x_i, if the two bits are equal, put a star (*) in the second (following) chromosome; otherwise, leave this bit without a change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one, and so on.

(b) For chromosomes in the second quarter (G2) of the population (from $M/4 + 1$ to $M/2$), apply the similarity operator (see Chapter 3).

Table 4. 3 The dynamic dissimilarity operator.

Before change: an example for the first quarter of chromosomes.

	m_1						m_2			
	R_1		$m_1 - R_1$				R_2	$m_2 - R_2$		
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	1	0	1	0	0	1	0	0	1	1
Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	1	0	1	*	0	1	1	*	*	1

After change: put randomly 0 or 1 in (*) bits

Ch. A	1	1	0	0	1	0	1	0	1	0
Ch. B	1	0	1	1	0	1	0	1	0	1

7) All chromosomes B created in this way replace the original ones in positions from 2 to $M/2$. New chromosomes are also generated in the way described at Step 5 on positions from $M/2+1$ to $M/2+M/4$. Then generate randomly chromosomes for the fourth group of the population. These will replace the fourth group of the chromosomes (on positions from $M/2+M/4+1$ to M).
8) Go to Step 2 and repeat until the stopping criterion is reached.

Notes:

a. We call the genetic operator performing the operations shown in Table 4.3 on a pair of chromosomes A and B the *dynamic dissimilarity operator*, and the genetic operator performing the operations shown in Table 3.2 the *similarity operator*.
b. The *dynamic schema operator* is shown in Table 4.2, it uses different sizes of fixed segments (gray color) and applies the *similarity operator* on the rest of chromosome.
c. The stopping criterion for the algorithm depends on the example being considered, see Section 4.5.

Figure 4. 1 shows the flowchart of the DSDSC algorithm.

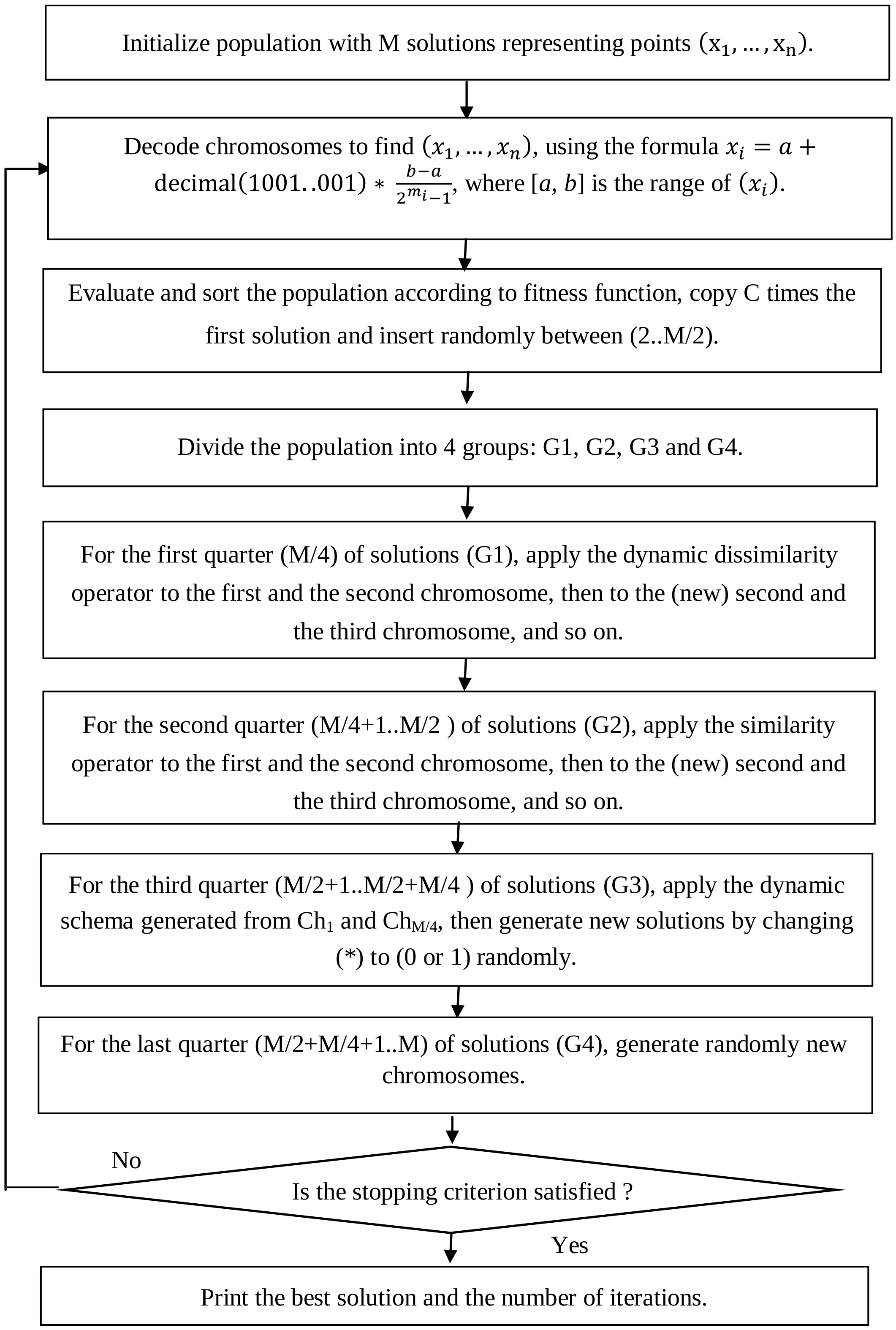

Initialize population with M solutions representing points $(x_1, \dots, x_n)$.
Decode chromosomes to find $(x_1, \dots, x_n)$, using the formula $x_i = a + \text{decimal}(1001..001) * \frac{b-a}{2^{m_i}-1}$, where $[a, b]$ is the range of (x_i).
Evaluate and sort the population according to fitness function, copy C times the first solution and insert randomly between (2..M/2).
Divide the population into 4 groups: G1, G2, G3 and G4.
For the first quarter (M/4) of solutions (G1), apply the dynamic dissimilarity operator to the first and the second chromosome, then to the (new) second and the third chromosome, and so on.
For the second quarter (M/4+1..M/2) of solutions (G2), apply the similarity operator to the first and the second chromosome, then to the (new) second and the third chromosome, and so on.
For the third quarter (M/2+1..M/2+M/4) of solutions (G3), apply the dynamic schema generated from Ch_1 and $Ch_{M/4}$, then generate new solutions by changing (*) to (0 or 1) randomly.
For the last quarter (M/2+M/4+1..M) of solutions (G4), generate randomly new chromosomes.
No
Is the stopping criterion satisfied ?
Yes
Print the best solution and the number of iterations.

Figure 4. 1 Flowchart of the DSDSC algorithm.

4.4 Schema analysis

A schema represents a number of similar strings, thus, a schema can be thought of as a representation of a certain region in the search space. The schema that represents the region containing the best solution must increase in the population to get the solutions in the best region [87], [9]. For example, assuming to have a part of the Zbigniew Michalewicz function $f(x_1, x_2) = 21.5 + x_1 \cdot sin(4\pi x_1) + x_2 \cdot sin(20\pi x_2)$, where $x_1, x_2 \in [0,1]$, as shown in Figure 4. 2, it is clear the maximum solution has $x_1 \in [0.6,0.8]$ in the region [0,1]. This function has many local maximum solutions of which only one is global, as shown in Figure 4. 2. Consider this region [0, 1] of x_1 represented by m bits $(1, \dots, m_1)$. Assume that we have two types of schemata: H0= (0 * * . . .*) representing the left region where $x_1 \in [0, 0.5]$, and H1= (1 * * . . . *) representing the right region, where $x_1 \in [0.5, 1]$. Since it is required to find a global optimum solution, it must be focused on schema H1 since it represents the region of a global solution. Also the same thing for x_2 [9] . However, it is possible that the region of a global optimal solution cannot be found this way. In such a case, the similarity operator and random generation of a part of chromosomes could help to find a better region.

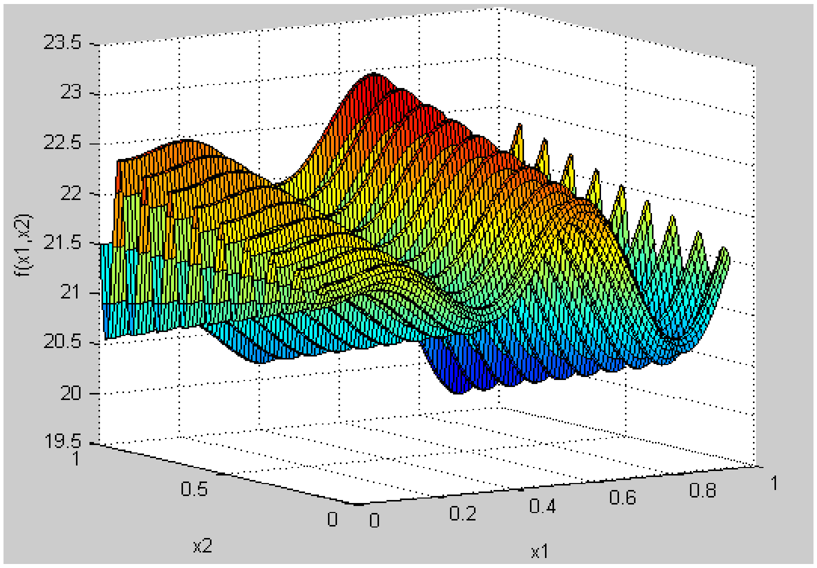


Figure 4. 2 A part of Michalewicz function.

4.5 Experimental results

In this section, we report on computational testing (by using the Matlab software) of the DSDSC algorithm on 22 test functions taken from literature:18 functions of 2 variables, one function of 4 variable and 5 functions of 10 variables. The result of DSDSC has been compared with the known global optimum and with the result of a classical GA taken from our experiments (Table 3.10), also, compared with CMA-ES and EA algorithms. The results are presented in Table 4.4, for 18 functions of 2 and in Table 4.5for 5 functions of 10 variables with one function of 4 variables, with the known optimal solutions mentioned in Appendix A. The algorithm with 80 chromosomes has been applied with the stopping criterion that the difference between the best solution and known optimal solution is less than the threshold specified in the second column (Tables 4.4, 4.5).

The DSDSC algorithm has found optimum solutions for some optimization problems (like Beale's, Schaffer n.2, Schwefel's,) that the classical genetic algorithm cannot reach to 100% success rate with bit string or double vector, as shown in Table 4.4, column nine. All success rates are 100% with 80 chromosomes for all problems.

The DSDSC algorithm keeps the best solution from each iteration at the first position until it is replaced by a better one.

Note that the maximum number of iterations to found the best solution was especially high (471) for the Rosenbrock’s valley function as shown in Table 4. 4. Also, the success rate for Michalewicz problem was 100% compared with the classical GA algorithm where it was 80% with the same number of chromosomes and generations. On the other hand, column three in Table 4. 4Table 4.4 shows the minimum number of iterations for finding an optimal solution was between 2 and 9 for all 18 test functions. Column five shows the average number of iterations for all successful runs. Table 4.5 shows the results for 10-dimensional functions with 160 chromosomes and the number of iterations fixed to 2000; both tables also show the run time (Min., Max., and Average).

Table 4. 4 The results for 50 runs of the DSDSC algorithm (80 chromosomes).

Function name	Threshold	Min number of iterations / Min time in seconds	Max number of iterations / Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Rate of success DSDSC	Rate of success GA
Easom	0.001	6	238	51	47.8	-0.9992	100%	100% DV
		0.0095	0.2297	0.0516				
Matyas	0.001	2	28	11	5.7	0.00040	100%	100% DV
		0.0043	0.0291	0.012				
Beale's	0.001	5	166	49	38.5	0.00047	100%	70% DV
		0.0078	0.1301	0.0568				
Booth's	0.001	4	65	20	16.4	0.00057	100%	100% DV
		0.0068	0.0576	0.0205				
Goldstein–Price	0.001	5	85	34	18.6	3.0004	100%	100% DV
		0.0074	0.0825	0.0364				
Schaffer N. 2	0.001	4	189	71	45.2	0.00028	100%	70% DV
		0.0072	0.1503	0.0747				
Schwefel's	0.001	6	282	41	49.4	0.00064	100%	0% BS
		0.0070	0.2284	0.0477				
Branins's rcos	0.001	5	203	28	42.4	0.39832	100%	100% DV
		0.0057	0.2461	0.0252				
Six-hump camel back	0.001	5	127	18	24.5	-1.0310	100%	100% DV
		0.0098	0.1347	0.0244				
Shubert	0.01	3	67	19	13.3	-186.719	100%	100% DV
		0.0044	0.0772	0.0208				
Martin and Gaddy	0.001	4	38	15	8.4	0.00043	100%	40% DV
		0.0082	0.0360	0.0169				
Michalewicz	0.04	9	280	67	57	38.8182	100%	80% DV
		0.0063	0.2123	0.0395				
Holder table	0.001	3	45	12	8	-19.208	100%	80% DV
		0.0057	0.0466	0.0188				
Drop-wave	0.001	7	172	48	36.8	-0.9996	100%	100%

		0.0094	0.2758	0.0539				BS
Levy N. 13	0.001	5	202	45	38.7	0.00044	100%	100% BS
		0.0069	0.2124	0.0486				
Rastrigin's	0.001	8	127	25	17.1	0.00037	100%	100% BS
		0.0127	0.0585	0.0203				
Sphere	0.001	3	19	7	8.2	0.000416	100%	100% BS
		0.0082	0.0173	0.0122				
Rosenbrock's valley	0.001	3	471	115	102.1	0.000535	100%	100% BS
		0.0104	0.1899	0.0536				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation. .

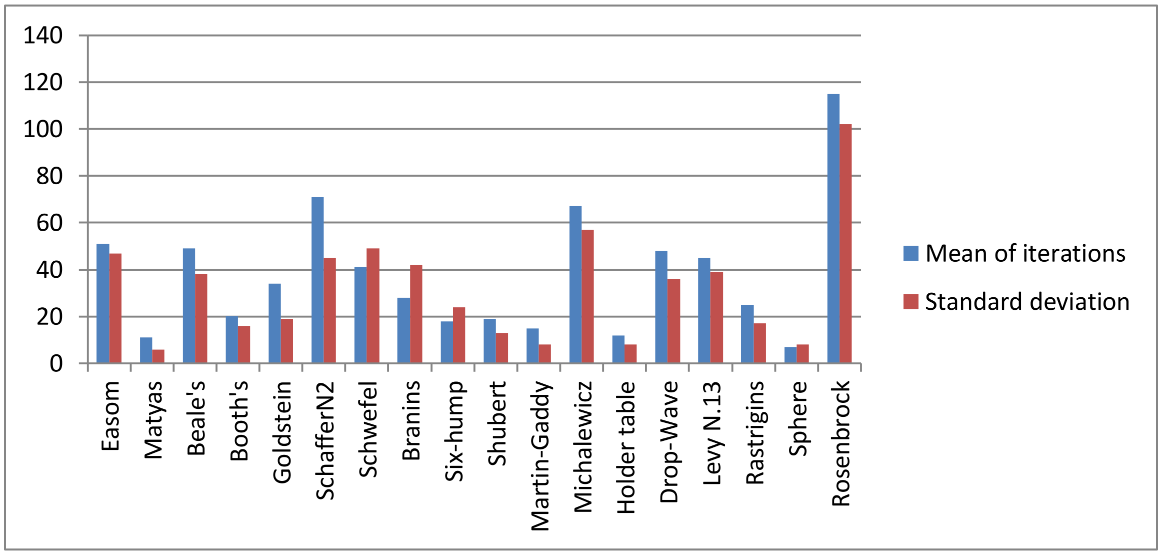
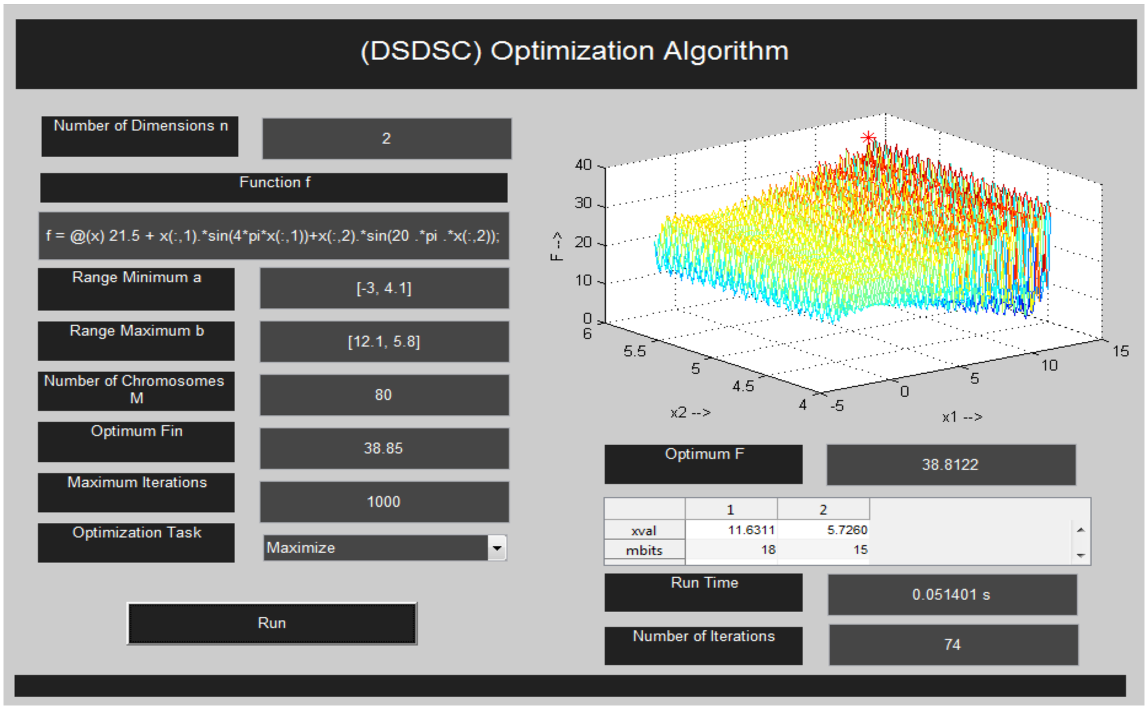


Figure 4. 3 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for DSDSC algorithm

Figure 4. 4 shows the GUI of DSDSC algorithm on Michalewicz function, Figure 4. 5 shows the GUI of DSDSC algorithm with Shubert function that has 18 optimum solutions.



(DSDSC) Optimization Algorithm

Number of Dimensions n: 2

Function f: f = @(x) 21.5 + x(:,1).*sin(4*pi*x(:,1))+x(:,2).*sin(20 .*pi .*x(:,2));

Range Minimum a: [-3, 4.1]

Range Maximum b: [12.1, 5.8]

Number of Chromosomes M: 80

Optimum Fin: 38.85

Maximum Iterations: 1000

Optimization Task: Maximize

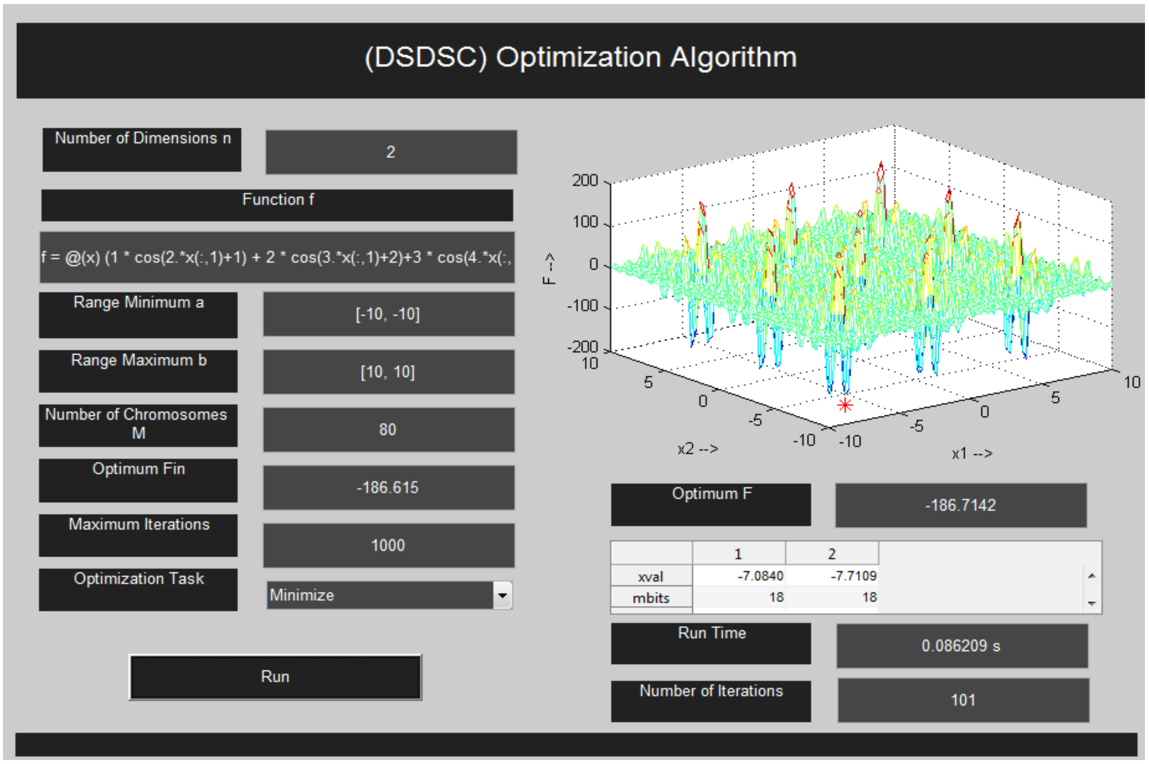
Optimum F: 38.8122

	1	2
xval	11.6311	5.7260
mbits	18	15

Run Time: 0.051401 s

Number of Iterations: 74

Figure 4. 4 The GUI of DSDSC algorithm



(DSDSC) Optimization Algorithm

Number of Dimensions n: 2

Function f: f = @(x) (1 * cos(2.*x(:,1)+1) + 2 * cos(3.*x(:,1)+2)+3 * cos(4.*x(:,

Range Minimum a: [-10, -10]

Range Maximum b: [10, 10]

Number of Chromosomes M: 80

Optimum Fin: -186.615

Maximum Iterations: 1000

Optimization Task: Minimize

Optimum F: -186.7142

	1	2
xval	-7.0840	-7.7109
mbits	18	18

Run Time: 0.086209 s

Number of Iterations: 101

Figure 4. 5 Shubert function with 18 optimum solutions

Table 4. 5 The results for 25 runs of the DSDSC algorithm for 10-dimensional functions with execution time and comparing with GA

Function name	Threshold	Min number of iterations	Max number of iterations	Mean no. of iterations for all successful runs	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Rate of success DSDSC	Rate of success GA
Sum Squares d=10	0.1	37	597	145	128.7	0.072731	100%	100% BS
		0.057789	0.89309	0.221053				
Sphere d=10	0.1	17	72	31	12.4	0.069036	100%	100% BS
		0.0251	0.1032	0.0464				
Sum of different powers d=10	0.1	1	5	3	1.2	0.073843	100%	100% BS
		0.0018	0.0092	0.0056				
Zakharov d=10	0.1	76	595	217	116	0.077189	100%	100% BS
		0.1056	0.7596	0.2883				
Rastrigin d=10	0.1	159	1978	1045	467	0.148285	92%	100% BS
		0.2105	2.5300	1.3786				
Ackley d=4	0.001	73	1706	644	532.2	0.000979	80%	100% BS
		0.0480	1.3318	0.6595				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

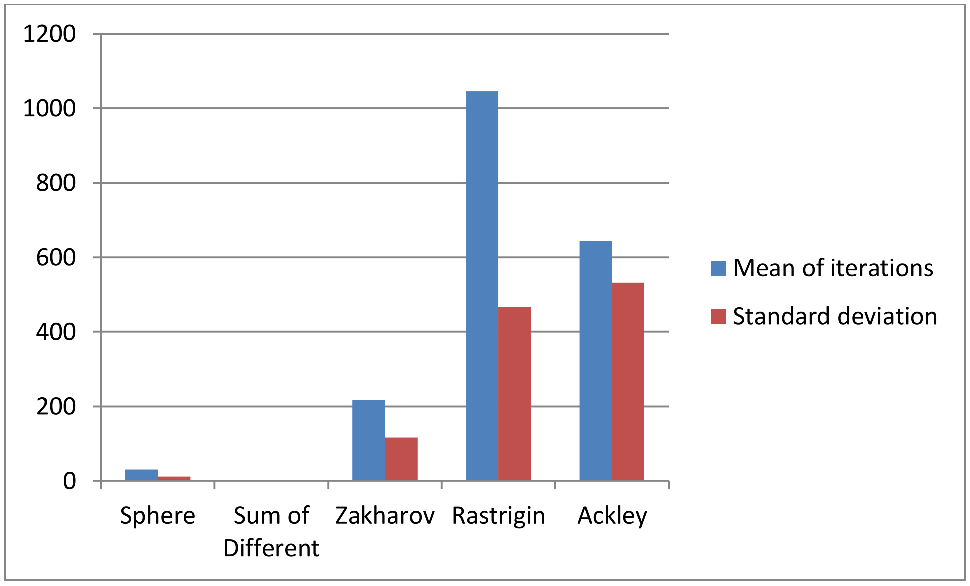


Figure 4. 6 The average number and standard deviation of iterations for 10-dimensional functions with 80 chromosomes for DSDSC algorithm

Table 4.6 presents a comparison of CMA-ES, DE and DSDSC algorithms in terms of mean number of function evaluations and success rate, by using 50 different runs, with 2500 maximum number of iterations and population size is 80 chromosomes.

Table 4. 6 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and DSDSC algorithms (50 runs, max 2500 iterations, 80 chromosomes).

function name	CMA-ES success rate	Function evaluations of CMA-ES	DE success rate	Function evaluations of DE	DSDSC success rate	Function evaluations of DSDSC
Easom	70%	17053	100%	3240	100%	4080
Matyas	100%	500	100%	2700	100%	880
Beale	100%	460	100%	3060	100%	3920
Booth's	100%	492	100%	2820	100%	1600

Goldstein–Price	100%	1812	100%	1620	100%	2720
Schaffer N.2	90%	6726	100%	5016	100%	5680
Schwefel's	0%	----	0%	----	**100%**	3280
Branins's rcos	100%	6876	100%	840	100%	2240
Six-hump camel	100%	780	100%	2160	100%	1440
Shubert	90%	2220	100%	8160	100%	1520
Martin and Gaddy	100%	1660	100%	2400	100%	1200
Michalewicz	100%	1848	0%	---	100%	5360
Drop-wave	50%	26470	94%	9048	100%	3840
Levy N. 13	100%	606	100%	1958	100%	3600
Rastrigin’s	80%	13134	100%	2388	100%	2000
Sphere	100%	720	100%	1800	100%	560
Ackley d=4	100%	2240	100%	3480	80%	90160
Rosenbrock’s	100%	1644	100%	4560	100%	9200

4.6 Conclusion

In this section, a new meta-heuristic optimization algorithm called DSDSC is introduced. DSDSC can be simply implemented, without too many parameters. It includes three genetic operators (the dynamic schema, dynamic dissimilarity and similarity operators), population sorting and random generation of a part of the population.

The experiments have shown quick convergence and the good global searching ability of the algorithm. The DSDSC algorithm is easy to understand and uses a simple classical representation of points in $\mathbb{R}^n$.

The DSDSC algorithm has only two parameters to be set by the user: the number M of chromosomes and R_i parameter in Step 5(a) in the algorithm. Therefore, it is easier to test than the classical GA where the user must try multiple runs to test different combinations of parameters. For all the examples, 80 chromosomes are enough to solve the problem. We see from Table 4.6 that the CMA-EA and DE algorithms did not find the solution for Schwefel's function, but DSDSC algorithm has found the solution in 100% of success rate.

CHAPTER FIVE: The DDS, FDS and MFDS Algorithms

5.1 Introduction

This chapter contains the following three algorithms:

1. Double Dynamic Schema with DSC algorithm (DDS Algorithm).
2. Free Dynamic Schema (FDS).
3. Multi Free Dynamic Schema (MFDS).

5.2 Double Dynamic Schema (DDS) algorithm

The idea of double population in evolutionary algorithms was used to improve the search for optimal solution also to increase the diversity of a population. In [97] the authors have used a double population with Swarm Optimization Algorithm for optimization problems, in [98] a dual-population genetic algorithm was presented, which employs two populations, where the main population was used to find a good solution to the given problem and the second population was used to evolve and provide controlled diversity to the main population.

In this section a new evolutionary algorithm for solving optimization problems called Double Dynamic Schema with Dissimilarity and Similarity of Chromosomes (DDS) is presented. This algorithm is complementary to our previous algorithms called Dynamic Schema with Dissimilarities and Similarities of Chromosomes (DSDSC) ([99] or Chapter 4) and Dissimilarity and Similarity of Chromosomes (DSC) ([100] or Chapter 3). In the DDS algorithm a new technique is used, that is, double population of chromosomes working together to improve the efficiency of optimization and increase the chance to reach the best solution, where the first population is the original one and the second one is a copy of the first one, but different types of operations are applied to it.

Briefly, the algorithm aims at finding the optimal solution by fixing the highest bits of a chromosome (i.e., fixing the highest bits of all variables $(x_1, \dots, x_n)$ which are contained in the chromosome) and changing the lower bits at the same time, thus the algorithm focuses on the searching in a small area that may contain the optimal solution.

5.2.1 Methodology

The DDS starts with a random population (P0) of M elements representing a number of solutions to the problem. This population is sorted, then a new population (P1) is formed which is a copy of a part of (P0), Each population (P0, P1) is divided into several equal groups and some different operators are applied to these groups (see Table 5. 1).

Table 5. 1 Populations (P0) and (P1) and the seven groups of chromosomes.

Original Groups of Chromosomes (P0)		**Copy Groups of Chromosomes (P1)**	
Ch_1	G1: To the first group the dynamic dissimilarity operator is applied.	Ch_1	G5: To the fifth group the dissimilarity operator is applied.
Ch_2		Ch_2	
Ch. …		Ch. …	
Ch. …		Ch. …	
$Ch_{M/4}$		$Ch_{M/4}$	
$Ch_{M/4+1}$	G2: To the second group the similarity operator is applied.	$Ch_{M/4+1}$	G6: To the sixth group the dynamic dissimilarity operator is applied.
$Ch_{M/4+2}$		$Ch_{M/4+2}$	
Ch. …		Ch. …	
Ch. …		Ch. …	
$Ch_{M/2}$		$Ch_{M/2}$	
$Ch_{M/2+1}$	G3: To the third group the dynamic schema operator is applied.	$Ch_{M/2+1}$	G7: To the seventh group the dynamic schema operator is applied.
$Ch_{M/2+2}$		$Ch_{M/2+2}$	
Ch. …		Ch. …	
Ch. …		Ch. …	
$Ch_{M/2+M/4}$		$Ch_{M/2+M/4}$	
$Ch_{M/2+M/4+1}$	G4: The fourth group is generated randomly.		
$Ch_{M/2+M/4+2}$			
Ch. …			
Ch. …			
Ch_M			

Briefly, the DDS creates new chromosomes by exploring dissimilarity, similarity, dynamic schema and dynamic dissimilarity. These operators are described as follows:

5.2.1.1 Dissimilarity operator

For the first two chromosomes (A,B) in a group, check all corresponding bits: if the two bits are equal, put a star (*) in the second (B) chromosome; otherwise leave this bit without change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third chromosome in the group, and so on (see Table 3. 1).

5.2.1.2 Similarity operator

For two chromosomes (A,B), check each corresponding bits: if the two bits are not equal, put a star (*) in the second (B) chromosome; otherwise leave this bit without change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one and so on (see Table 3.2).

5.2.1.3 Dynamic schema operator

The dynamic schema operator is applied onto two chromosomes (A, B). This operator works as follows (see Table 4. 2):

First, divide each chromosome into n parts corresponding to variables $(x_1, \ldots, x_n)$, the i-th part having length m_i, where m_i is the number of bits for x_i. Next, for each variable x_i, generate a random integer R_i from the set $\{3, \ldots, m_i/2\}$. Define the “gray” part of x_i as the first segment of length R_i of the string corresponding to x_i. Define the “white” part of x_i as the second segment of length $m_i - R_i$ of the same string.

For the “white” parts of both chromosomes, if the two bits are not equal, put a star (*) in the schema; otherwise leave this bit without change in the schema. After finding the schema, copy it $K = M/4$ times and put it in group (G3), then put randomly 0 or 1 in the positions having (*). The positions marked in “gray” are kept unchanged.

Note. The name "dynamic schema operator" is justified by the fact that the lengths of "gray" and "white" segments of chromosomes may vary from iteration to iteration (see Table 4.2).

5.2.1.4 Dynamic dissimilarity operator

The dynamic dissimilarity operator is applied onto two chromosomes (A, B). This operator works similarly to the dynamic schema operator only to find the "gray" and "white" parts corresponding to variables $(x_1, \ldots, x_n)$. The "gray" part of x_i is not destroyed, in the "white" part of x_i, if the two bits are equal, put a star (*) in the second (B) chromosome; otherwise, leave this bit without change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*) in the second chromosome. Compare this new second chromosome with the third one in the same way, and so on (see Table 4.2).

5.2.2 The DDS algorithm

The following optimization problem is considered:

$$\text{f}: \mathbb{R}^{\text{n}} \longrightarrow \mathbb{R}$$

$$\text{minimize}|\text{maximize } f(x_1, \ldots, x_n) \text{ subject to}$$

$$x_i \in [a_i, b_i], i = 1, \ldots, n$$

where $f: \mathbb{R}^n \longrightarrow \mathbb{R}$ is a given function.

In the algorithm described below, the encoding of chromosomes is the same as in Chapter 3.

Let M be a positive integer divisible by 8. The DDS algorithm consists of the following steps:

1. Generate $2M - M/4$ chromosomes, each chromosome representing a point $(x_1, \ldots, x_n)$. Divide the chromosomes into two populations (P0) and (P1), where (P0)

consists of four groups (G1, G2, G3, G4), and (P1) consists of three groups (G5, G6, G7), each group having M/4 chromosomes.

2. Compute the values of the fitness function f for each chromosome in the population (G1,…,G7).
3. Sort the chromosomes according to the descending (for maximization) or ascending (for minimization) values of the fitness function.
4. Copy the groups (G1, G2) onto (G5, G6), replacing the original chromosomes.
5. Copy C times the first chromosome and put it in C randomly chosen positions in the first half of population (P0), replacing the original chromosomes, where $C = M/8$.
6. Apply the dynamic schema operator for chromosomes A = Ch_1 and B = $Ch_{M/4}$ from populations (P0), (that is, the chromosomes on positions 1 and M/4, respectively). Copy this schema M/4 times and put it in (G3).
7. Apply the dynamic schema operator for chromosomes A = Ch_1 and B = $Ch_{M/4}$ from populations (P0), (that is, the chromosomes on positions 1 and $M/4$ respectively). Copy this schema $M/4$ times and put it in (G7).
8. Apply the dynamic dissimilarity and similarity operators to groups (G1) and (G2) respectively. Apply the dissimilarity and dynamic dissimilarity operators to groups (G5) and (G6) respectively.
9. All the chromosomes created in Steps 6 to 8 replace the original ones in positions from 2 to $3M/4$ in populations (P0) and (P1). Then randomly generate chromosomes for group (G4).
10. Go to Step 2 and repeat until the stopping criterion is reached.

Note:

- The stopping criterion for the algorithm depends on the example being considered, see Section 5.2.3.

In Figure 5.1 we show the DDS algorithm flowchart.

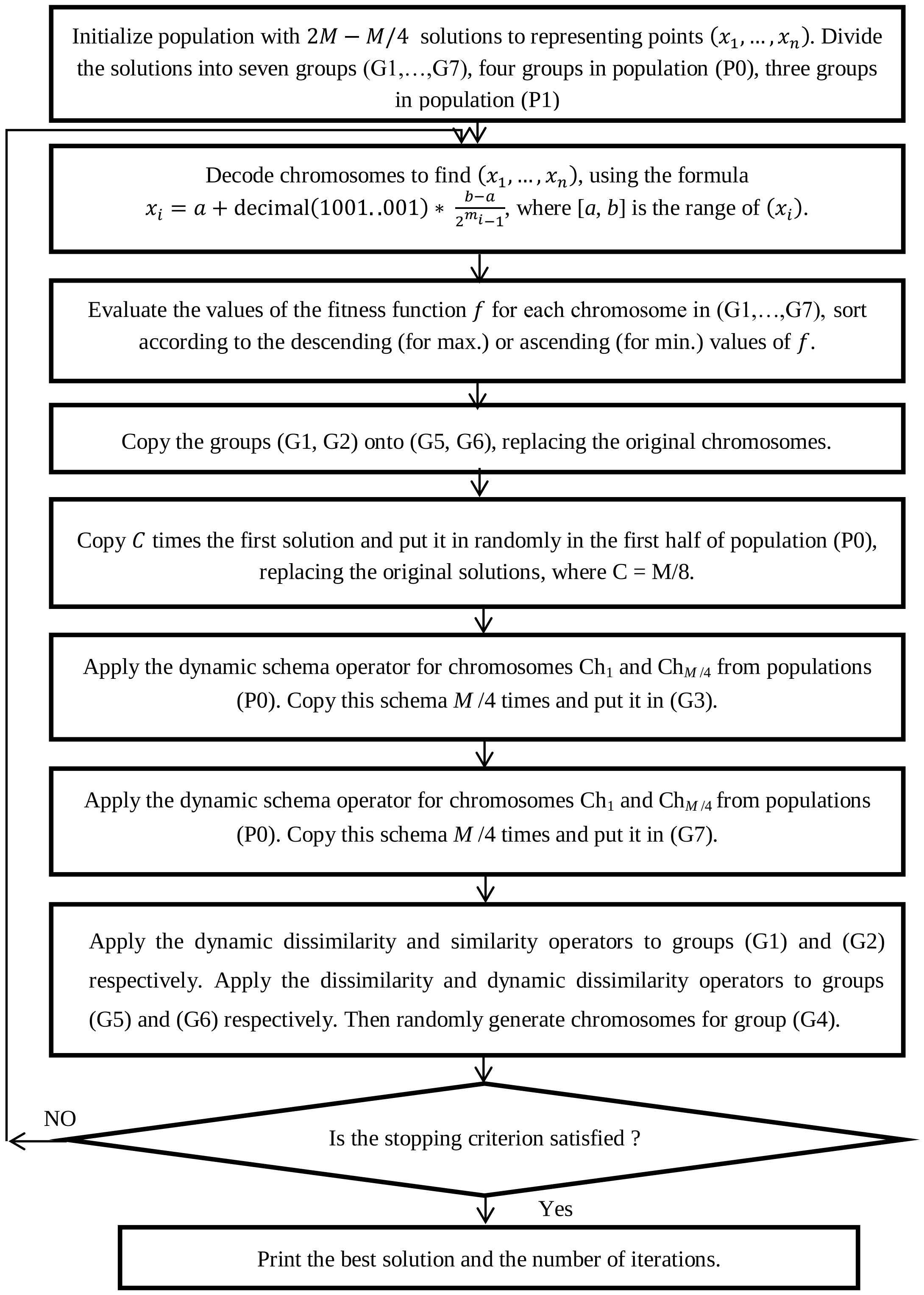

Initialize population with $2M - M/4$ solutions to representing points $(x_1, \dots, x_n)$. Divide the solutions into seven groups (G1,…,G7), four groups in population (P0), three groups in population (P1)
Decode chromosomes to find $(x_1, \dots, x_n)$, using the formula $x_i = a + \text{decimal}(1001..001) * \frac{b-a}{2^{m_i-1}}$, where [$a$, b] is the range of (x_i).
Evaluate the values of the fitness function f for each chromosome in (G1,…,G7), sort according to the descending (for max.) or ascending (for min.) values of f.
Copy the groups (G1, G2) onto (G5, G6), replacing the original chromosomes.
Copy C times the first solution and put it in randomly in the first half of population (P0), replacing the original solutions, where C = M/8.
Apply the dynamic schema operator for chromosomes Ch_1 and $\text{Ch}_{M/4}$ from populations (P0). Copy this schema M /4 times and put it in (G3).
Apply the dynamic schema operator for chromosomes Ch_1 and $\text{Ch}_{M/4}$ from populations (P0). Copy this schema M /4 times and put it in (G7).
Apply the dynamic dissimilarity and similarity operators to groups (G1) and (G2) respectively. Apply the dissimilarity and dynamic dissimilarity operators to groups (G5) and (G6) respectively. Then randomly generate chromosomes for group (G4).
Is the stopping criterion satisfied ?
NO
Yes
Print the best solution and the number of iterations.

Figure 5. 1 Flowchart of the DDS algorithm.

5.2.3 Experimental results

In this section, we report on computational testing (by using the Matlab R2015b software on a computer having CPU core i5 2.4 MHz, 8 GB RAM) of the DDS algorithm on 18 functions of 2 variables, one function of 4 variable and 5 functions of 10 variables. The test functions are taken from literature. After each test, the result of DDS has been compared with the known global optimum and with the result of a classical GA taken from our experiments (Table 3. 10), also, compared with CMA-ES and EA algorithms. All 22 tested functions with optimal solutions are mentioned in Appendix A. We have applied the algorithm with 80 chromosomes (P0) with the stopping criterion that the difference between our best solution and the known optimal solution is less than or equal to a given threshold. This threshold was equal to 0.001 for most two-dimentional functions, 0.01 for the Shubert function, 0.04 for the Michalewicz function, and 0.1 for ten-dimensional functions.

The DDS algorithm has found optimum solutions for some optimization problems (like Beale's, Schaffer n.2, Schwefel's,) that the classical genetic algorithm cannot reach to 100% success rate with bit string or double vector, as shown in Table 5. 2, column nine. For our algorithm all success rates are 100% with 80 chromosomes in (P0) for all problems.

The DDS algorithm keeps the best solution from each iteration at the first position until it is replaced by a better one.

Note that the average number of iterations to find the best solution was especially high (71) for the Michalewicz function, see Table 5.2. For 10-dimenisional problems we used 160 chromosomes for population (P0) with maximum 2000 iterations. Table 5. 3 shows the minimum, maximum and average numbers of function evaluations for 25 runs of the DDS algorithm. Table 5.4 shows a comparison of CMA-ES, DE and DDS algorithms in terms of mean number of function evaluations and success rate.

Table 5.5shows the number of function evaluations for 50 runs of the DDS algorithm for all functions.

Figures 5.2. 5.3 present the average number of iterations with standard deviation of iterations for 2-dimensional and 10-dimensional functions respectively for DDS algorithm.

Table 5. 2 The results for 50 runs of the DDS algorithm with run time (80 chromosomes).

Function name	Min number of iterations / Min time in seconds	Max number of iterations / Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution for all successful runs	Success rate of DSC And DSDSC	Success rate of DDS	Rate of success GA
Easom	4	291	62	71	-0.9993	100%	100%	100% DV
	0.0080	0.3147	0.0685					
Matyas	2	10	5	1.4	0.00048	100%	100%	100% DV
	0.0053	0.0202	0.0089					
Beale's	2	74	16	16.6	0.00049	100%	100%	70% DV
	0.0055	0.0816	0.0203					
Booth's	2	48	17	11.7	0.00051	100%	100%	100% DV
	0.0054	0.0504	0.0208					
Goldstein–Price	2	62	20	13.3	3.00049	100%	100%	100% DV
	0.0091	0.0678	0.0248					
Schaffer N.2	2	39	14	8.2	0.00035	100%	100%	70% DV
	0.0055	0.0478	0.0186					
Schwefel's	8	253	65	56.7	0.00068	100%	100%	0% BS
	0.0120	0.2724	0.0726					
Branins's rcos	2	103	9	15.4	0.39841	100%	100%	100% DV
	0.0052	0.1095	0.01368					
Six-hump camel back	2	61	8	9.8	-1.0311	100%	100%	100% DV
	0.0053	0.0670	0.0125					
Shubert	2	169	33	34	-186.714	100%	100%	100% DV
	0.0058	0.1928	0.0421					
Martin and Gaddy	2	11	6	1.8	0.00044	100%	100%	40% DV
	0.0054	0.0146	0.0097					

Michalewicz	2	546	71	103	38.8184	100%	100%	80% DV
	0.0055	0.5291	0.0721					
Holder table	4	87	24	19.4	-19.208	100%	100%	80% DV
	0.0089	0.0979	0.0297					
Drop-wave	6	189	44	40	-0.9995	100%	100%	100% BS
	0.0099	0.1982	0.0493					
Levy N. 13	4	47	19	11	0.00052	100%	100%	100% BS
	0.0080	0.0685	0.0248					
Rastrigin's	8	133	38	29.4	0.00041	100%	100%	100% BS
	0.0130	0.1539	0.0492					
Sphere	2	10	4	2	0.00334	100%	100%	100% BS
	0.0059	0.0184	0.0125					
Rosenbrock's valley	3	102	24	32.7	0.00055	100%	100%	100% BS
	0.0059	0.9985	0.0307					

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

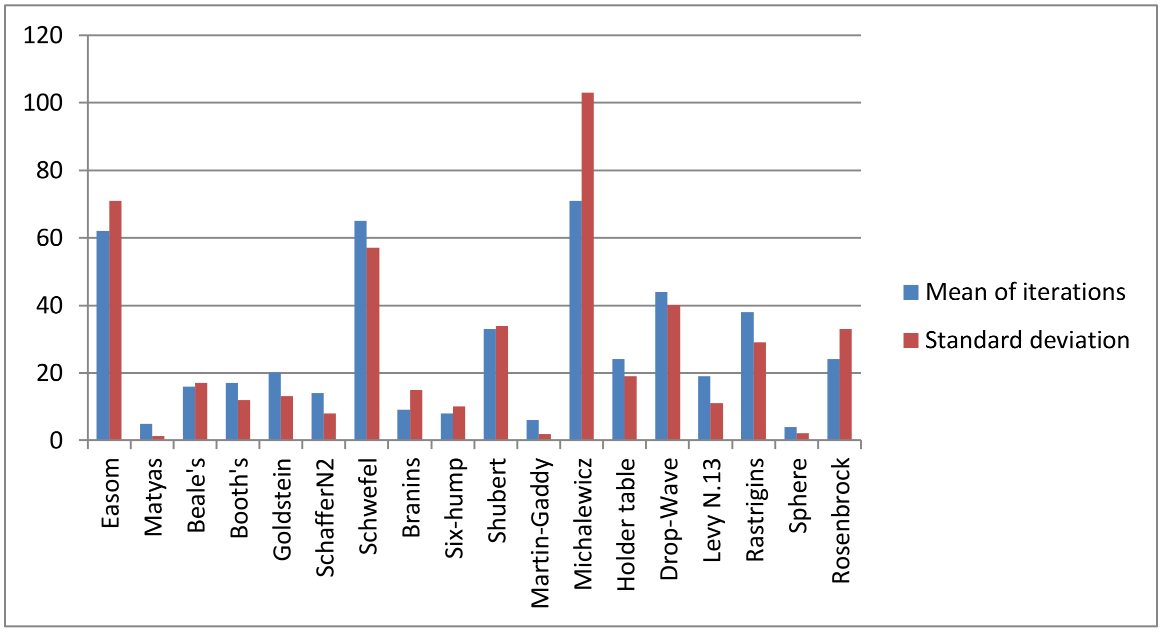


Figure 5. 2 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for DDS algorithm

Table 5. 3 The results for 25 runs of the DDS algorithm for 10-dimensional functions with execution time (160 chromosomes).

Function name	Min number of iterations / Min time in seconds	Max number of iterations/ Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution for all successful runs	Success rate of DSC And DSDSC	Success rate of DDS	Success rate of GA
Sum Squares d=10	38	251	128	70.7	0.07277	100%	25%	100% BS
	0.1007	0.6413	0.3354					
Sphere d=10	14	38	23	18	0.07304	100%	100%	100% BS
	0.0351	0.0954	0.0582					
Sum of different powers d=10	1	7	4	1.5	0.02955	100%	100%	100% BS
	0.0029	0.0210	0.0113					
Zakharov d=10	18	1691	468	378	0.32031	12% DSC 100% DSDSC	80%	100% BS
	0.0495	4.2347	1.6494					
Rastrigin d=10	2000	2000	2000	0	24.333	0% DSC 92% DSDSC	0%	100% BS
	4.3092	4.5206	4.3889					
Ackley d=4	57	1767	802	635	0.02594	100%	50%	100% BS
	0.07358	3.2348	2.5438					

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

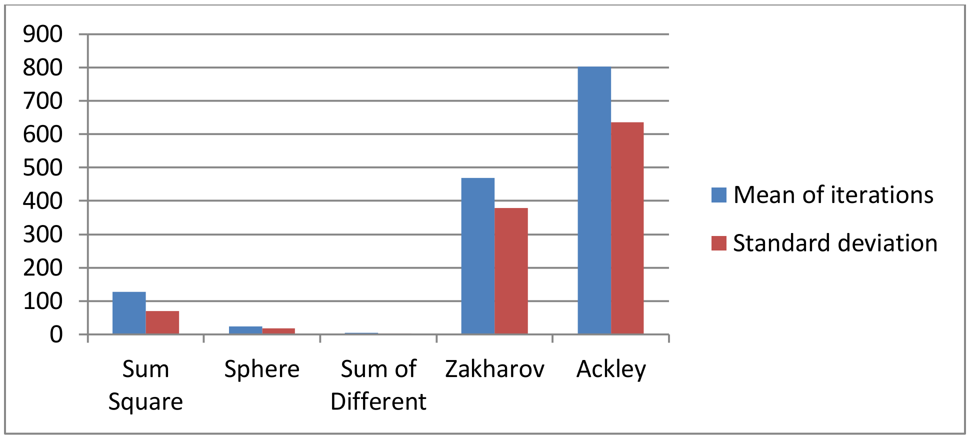


Figure 5. 3 The average number and standard deviation of iterations for 10-dimensional functions with 160 chromosomes for DDS algorithm

Table 5. 4 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and DDS algorithms (50 runs, max 2500 iterations, 80 chromosomes)

function name	CMA-ES success rate	Function evaluations of CMA-ES	DE success rate	Function evaluations of DE	DDS success rate	Function evaluations of DDS
Easom	70%	17053	100%	3240	100%	8680
Matyas	100%	500	100%	2700	100%	700
Beale	100%	460	100%	3060	100%	2240
Booth's	100%	492	100%	2820	100%	2380
Goldstein–Price	100%	1812	100%	1620	100%	2800
Schaffer N.2	90%	6726	100%	5016	100%	1960
Schwefel's	0%	----	0%	----	**100%**	9100
Branins's rcos	100%	6876	100%	840	100%	1260

Six-hump camel	100%	780	100%	2160	100%	1120
Shubert	90%	2220	100%	8160	100%	4620
Martin and Gaddy	100%	1660	100%	2400	100%	840
Michalewicz	100%	1848	0%	---	100%	9940
Drop-wave	50%	26470	94%	9048	100%	6160
Levy N. 13	100%	606	100%	1958	100%	2660
Rastrigin's	80%	13134	100%	2388	100%	5320
Sphere	100%	720	100%	1800	100%	560
Rosenbrock's valley	100%	1644	100%	4560	100%	3360

Table 5. 5 The number of function evaluations for 50 runs of the DDS algorithm

Function name	**Min No. of function evaluations**	**Max No. of function evaluations**	**Average No. of function evaluations**
Easom	560	40740	8680
Matyas	280	1400	700
Beale's	280	10360	2240
Booth's	280	6720	2380
Goldstein–Price	280	8680	2800
Schaffer N.2	280	5460	1960
Schwefel's	1120	35420	9100
Branins's rcos	280	14420	1260

Six-hump camel back	280	8540	1120
Shubert	280	23660	4620
Martin and Gaddy	280	1540	840
Michalewicz	280	76440	9940
Holder table	560	12180	3360
Drop-wave	840	26460	6160
Levy N. 13	560	6580	2660
Rastrigin's	1120	18620	5320
Rosenbrock	420	14280	3360
Sum Squares 10-D	10640	70280	35840
Sphere 10-D	3920	10640	6440
Sum of different powers 10-D	280	1960	1120
Zakharov 10-D	22400	475160	131040
Rastrigin 10-D	***	***	***

In Table 5.6 we compare the average number of function evaluations among the DSC, DSDSC and DDS algorithms, also Figure 5.4 presents the values of Table 5.6. It is clear that the DDS algorithm has the best values for most tested functions.

Table 5.7 presents a comparison of the success rate and the number of function evaluations (for two-dimensional functions only) for three algorithms: Bees Algorithm (BA), Particle Swarm Optimization (PSO), and DDS. The results for BA and PSO are taken from [101].

Table 5. 6 Comparing the average numbers of function evaluations for 50 runs of the DSC, DSDSC and DDS algorithms.

Function name	DSC	DSDSC	DDS
Eesom	7040	4080	8680
Matyas	2480	880	700
Beale's	7440	3920	2240
Booth's	12080	1600	2380
Goldstein–Price	10720	2720	2800
Schaffer N.2	22240	5680	1960
Schwefel's	44880	3280	9100
Branins's rcos	6880	2240	1260
Six-hump camel back	3120	1440	1120
Shubert	2560	1520	4620
Martin and Gaddy	2880	1200	840
Michalewicz	16560	5360	9940
Holder table	3760	960	3360
Drop-wave	15520	3840	6160
Levy N. 13	23200	3600	2660
Rastrignins	5680	2000	5320
Rosenbrock’s valley	8080	9200	3360

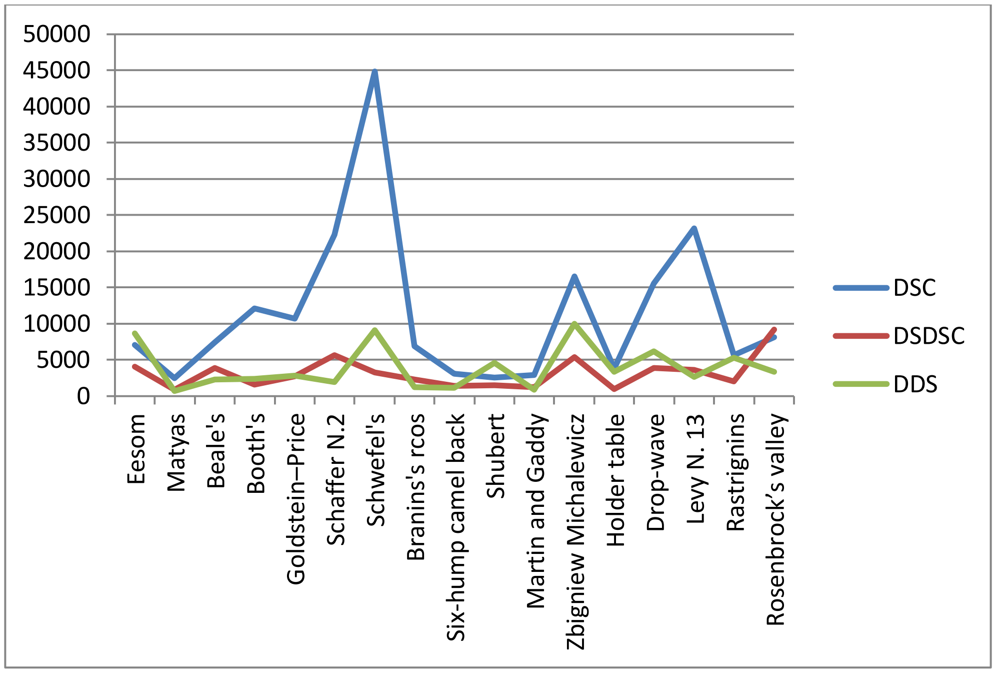


Figure 5. 4 Comparing the average numbers of function evaluations for DSC, DSDSC and DDS algorithms.

Table 5. 7 Comparing the average number of functions evaluations and success rate of BA, PSO and DDS algorithms

Function name	**BA**	**Fun. Eval. of BA**	**PSO**	**Fun. Eval. of PSO**	**DDS**	**Fun. Eval. of DDS**
Easom	72%	5868	100%	2094	100%	8680
Shubert	0%	---	100%	3046	100%	4620
Schwefel's	85%	5385	86%	3622	100%	9100
Goldstein–Price	7%	9628	100%	1465	100%	2800
Martin and Gaddy	100%	1448	3%	9707	100%	840
Rosenbrock	46%	7197	100%	1407	100%	3360

5.2.3 Conclusion

The DDS is a new multi-population evolutionary algorithm that uses two populations. This algorithm uses different operators to find the optimal solution, where through the dynamic schema operator the algorithm obtains the best area of solutions, and searches within that area in each iteration as it detects the schema from the best solution in the population. The dynamic dissimilarity operator performs searching in a wide range of solutions in (G1) and (G6), where the high bits are kept without change and the lower bits are changed. The dissimilarity and similarity operators possess the ability of searching in the whole search space because every bit of a chromosome can be changed by them. The fifth operator generates chromosomes randomly in (G4) to help increasing the diversity and not to stick in a local optimum solution.

We have applied the GA, DSC, DSDSC, DDS algorithms on 22 test functions taken from literature (Appendix A) with 2 and 10 dimensions. The results show the DDS algorithm is superior on the GA and DSC and DSDSC algorithms for most two-dimensions functions.

Through our experiments we found that whenever the function range is small like (-1, 1) or (-5, 5), the solution was obtained faster compared to the larger range (-500,500).

5.3 Free Dynamic Schema Algorithm (FDS)

This algorithm is very similar to DDS algorithm (see Section 5.2 or [102]). The only change is that the dynamic schema operator (applied to G3 and G7) is now replaced by the free dynamic schema operator in which the schema is found from the first chromosome only, as explained in Table 5.8.

Table 5. 8 The free dynamic schema operator.

Before change: an example for finding schema from the first chromosome.

Here shadow bits are not destroyed.

No. of Ch.	m_1						m_2			
	R_1		$m_1 - R_1$				R_2	$m_2 - R_2$		
Ch_1	1	1	0	0	1	0	1	0	1	0
Schema	1	1	*	*	*	*	1	*	*	*

After finding the schema: put it in $M/2$....$M/2 + M/4$ positions.

$Ch_{M/2+1}$	1	1	*	*	*	*	1	*	*	*
$Ch_{M/2+2}$	1	1	*	*	*	*	1	*	*	*
Ch. …	1	1	*	*	*	*	1	*	*	*
Ch. …	1	1	*	*	*	*	1	*	*	*
$Ch_{M/2+M/4}$	1	1	*	*	*	*	1	*	*	*

After change: put randomly 0 or 1 in (*) bits.

$Ch_{M/2+1}$	1	1	1	1	1	0	1	0	0	1
$Ch_{M/2+2}$	1	1	1	0	0	0	1	1	1	1
Ch. …	1	1	0	1	1	0	1	1	1	0
Ch. …	1	1	0	1	0	1	1	0	0	0
$Ch_{M/2+M/4}$	1	1	1	0	1	1	1	0	1	1

Suppose f is a one-dimensional function with range [0,1], as shown in Figure 5.5, This function is represented by binary representation consisting of four bits, (0000,0001,…,1111), that means the range is divided into 16 segments.

The principle of free schema is as follows: Suppose there is a solution 0100 colored in gray, if the free schema operator is applied, for example with $R_1 = 2$, then bits number 1 and 2 are not changed, but in bits 3 and 4 we put *s in the discovered schema (01**). Then we randomly put 0 or 1 in positions having *s. Here the schema will cover all the subspace colored with green, in the same way another schema (10**) will cover all the subspace colored in red, as shown in Figure 5.5.

Bits:	1234
Ch.1:	0100
Schema:	01**
Sol.1:	0111
Sol.2:	0110
Sol.3:	0101

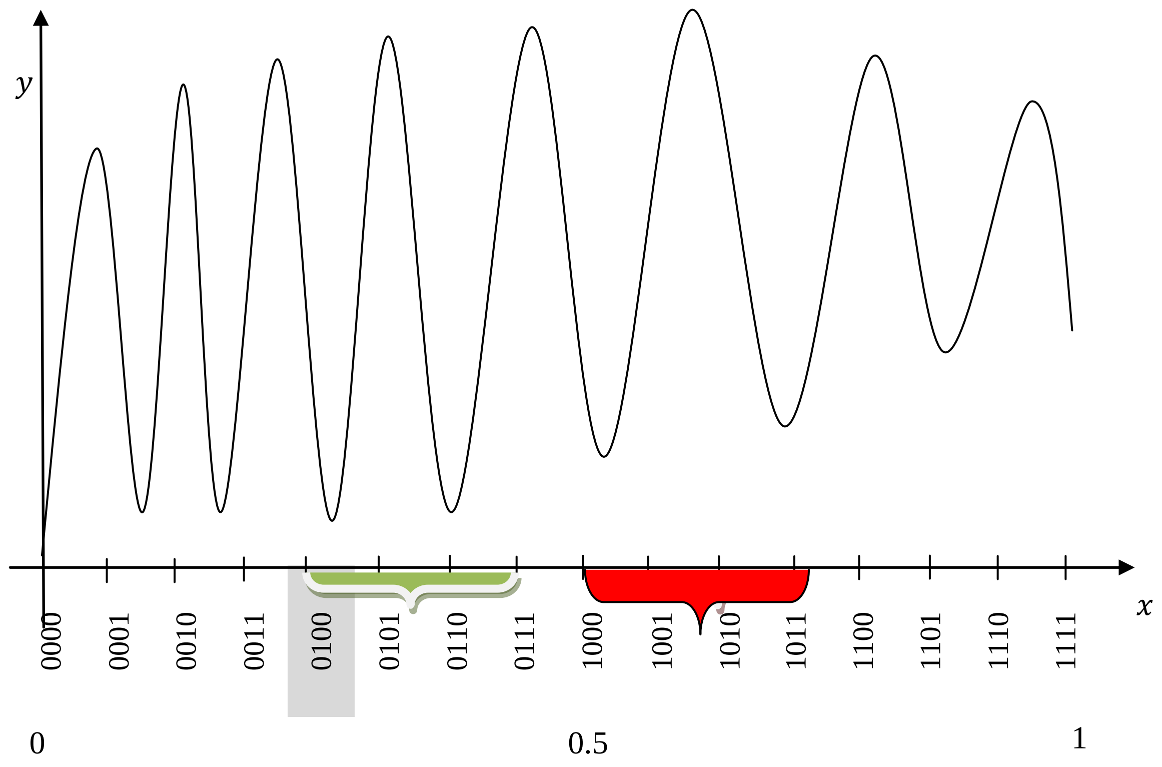


Figure 5. 5 Free dynamic schema operator.

5.3.1 Experimental results

In this section, we report on computational testing of the FDS algorithm on 18 functions of 2 variables, one function of 4 variable and 5 functions of 10 variables. After each test, the result of FDS has been compared with the known global optimum and with the result of a CGA taken from our experimental result (see Table 3.10), also, in Table

5.11 a comparison of the mean number of function evaluations and success rate of CMA-ES, DE and FDS algorithms is presented. All 22 tested functions with optimal solutions are mentioned in Appendix A. We have applied the algorithm with 80 chromosomes (P0) with the stopping criterion that the difference between our best solution and the known optimal solution is less than or equal to a given threshold.

The FDS algorithm has found optimum solutions for some optimization problems (like Beale's, Schaffer n.2, Schwefel's,) that the classical genetic algorithm cannot reach with 100% success rate with bit string or double vector, as shown in Table 5. 9, column nine. For our algorithm all success rates are 100% with 80 chromosomes in (P0) for all problems. Table 5. 10 shows the minimum, maximum and average numbers of iterations with standard deviation of iterations and comparison with GA for 25 runs of the FDS algorithm for 10-dimensional functions with 160 chromosomes in (P0).

Figures 5.6, 5.7 present the average number of iterations with standard deviation of iterations for 2-dimensional and 10-dimensional functions respectively for the FDS algorithm.

The FDS algorithm keeps the best solution from each iteration at the first position until it is replaced by a better one.

Table 5. 9 The results for 50 runs of the FDS algorithm.

Function name	Threshold	Min number of iteration/ Min time in seconds	Max number of iterations/ Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Success rate of FDS	Rate of success GA
Easom	0.001	24	241	89	51.9	-0.99936	100%	100% DV
		0.02824	0.26588	0.1024				
Matyas	0.001	2	14	6	2.6	0.000475	100%	100% DV
		0.00542	0.01726	0.0101				
Beale's	0.001	2	18	8	4	0.0004	100%	70%

		0.00555	0.02215	0.0123		99		DV
Booth's	0.001	3	37	12	6.7	0.000537	100%	100% DV
		0.00672	0.03861	0.0152				
Goldstein–Price	0.001	10	115	35	21.6	3.000507	100%	100% DV
		0.01423	0.12671	0.0409				
Schaffer N.2	0.001	2	58	16	11.7	0.00036	100%	70% DV
		0.00588	0.06565	0.0204				
Schwefel's	0.001	11	130	39	27.5	0.000662	100%	0% BS
		0.01554	0.13659	0.046				
Branins's rcos	0.001	2	89	11	15.8	0.398425	100%	100% DV
		0.00496	0.12197	0.0171				
Six-hump camel back	0.001	2	27	7	5.3	-1.03111	100%	100% DV
		0.00550	0.03045	0.0110				
Shubert	0.01	3	134	45	50.4	-186.717	100%	100% DV
		0.00445	0.15403	0.0562				
Martin and Gaddy	0.001	2	12	5	2.5	0.000471	100%	40% DV
		0.00171	0.01534	0.0093				
Michalewicz	0.04	3	166	55	37.8	38.81536	100%	80% DV
		0.00902	0.17056	0.0605				
Holder table	0.001	4	55	19	12.5	-19.208	100%	80% DV
		0.007824	0.062127	0.0239				
Drop-wave	0.001	7	111	45	23.5	-0.99952	100%	100% BS
		0.011624	0.122926	0.0502				
Levy N. 13	0.001	4	63	19	11.2	0.000541	100%	100% BS
		0.007922	0.066639	0.0235				
Rastrigin’s	0.001	11	131	58	31.4	0.000392646	100%	100% BS
		0.0199	0.1079	0.0555				
Sphere	0.001	2	15	7	3.5	0.00046	100%	100% BS
		0.00693	0.0222	0.0161				

Rosenbrock's valley	0.001	4	56	18	13.2	0.000623	100%	100% BS
		0.0088	0.0538	0.0255				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

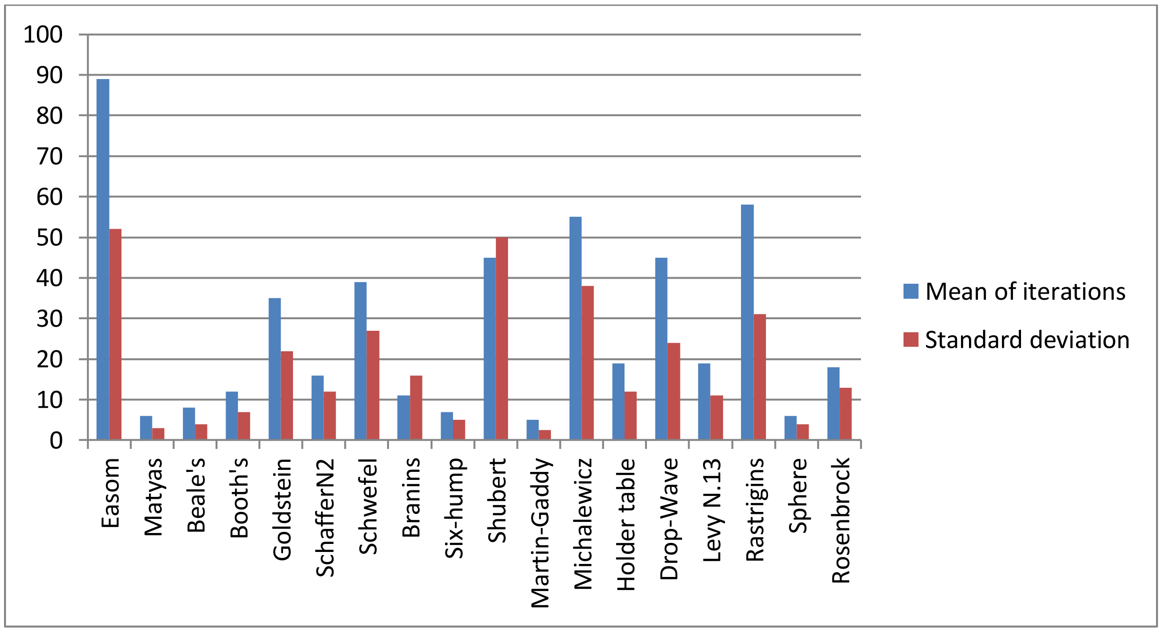


Figure 5. 6 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for FDS algorithm

Table 5. 10 The results for 25 runs of the FDS algorithm for 10-dimensional functions with run time.

Function name	Threshold	Min number of iterations / Min time in seconds	Max number of iterations / Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Success rate of FDS	Success rate of GA
Sum Squares d=10	0.1	164	634	320	141	0.082916	100%	100% BS
		0.3055	1.1877	0.6115				
Sphere	0.1	29	102	51	17	0.083032	100%	100% BS

d=10		0.0534	0.1781	0.0928				
Sum of different powers d=10	0.1	2	9	4	2.1	0.03835	100%	100% BS
		0.0032	0.0599	0.0115				
Zakharov d=10	0.1	180	1711	581	395	0.087195	100%	100% BS
		0.4853	4.4823	1.5508				
Rastrigin d=10	0.1	680	1953	1159	454.4	1.490886	32% 84%*	100% BS
		2.1410	6.1124	3.9758				
Ackley d=4	0.001	111	1334	536	465	0.001406	86%	100% BS
		0.1133	2.4283	1.3704				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

*we found this result by changing the size of R_i to a random number from {0, 1,.., m_i}, with 200 chromosomes and 2000 iterations.

By comparing the results for the DDS and FDS algorithms, we can see that the FDS is better than DDS for two-dimensional functions, while DDS is better than FDS for ten-dimensional functions.

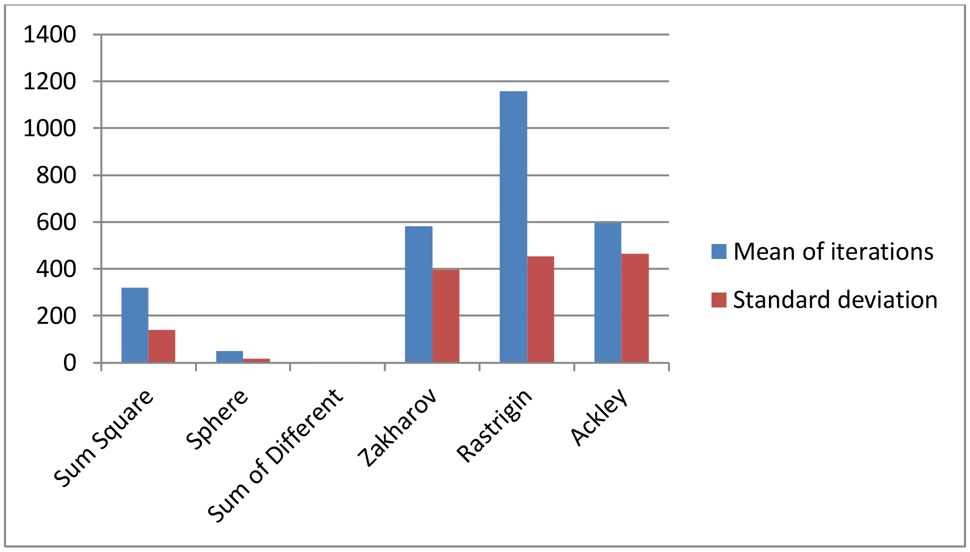


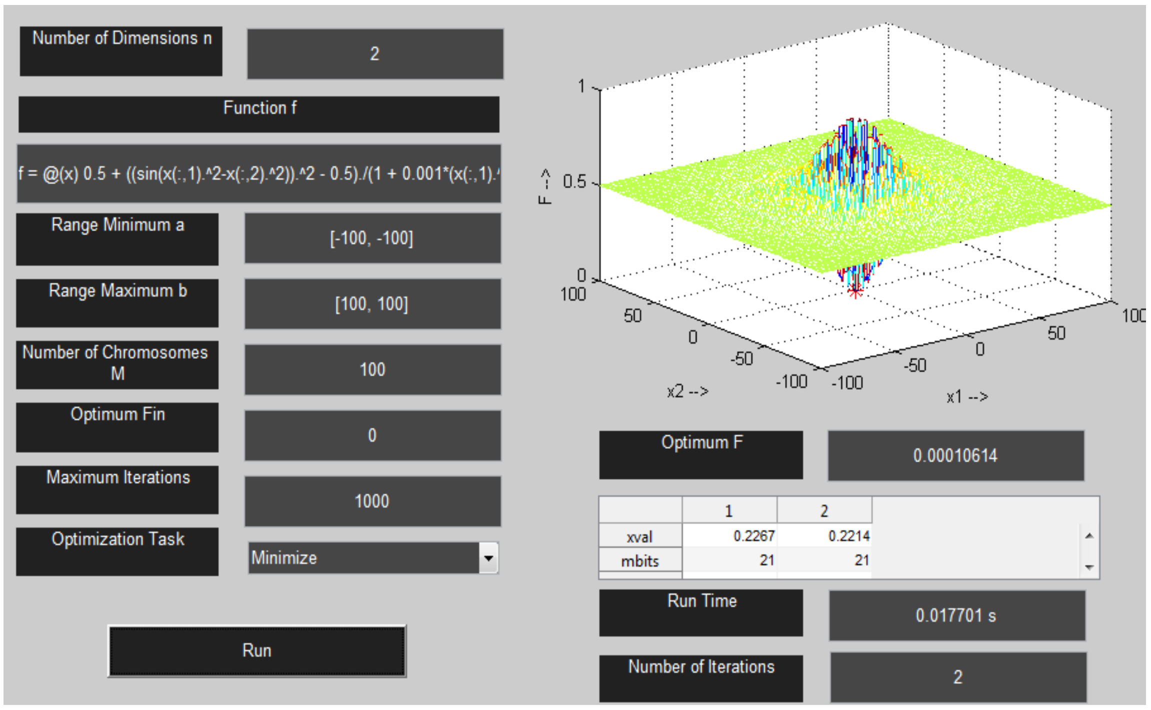
Figure 5. 7 The average number and standard deviation of iterations for 10-dimensional functions with 160 chromosomes for FDS algorithm

Table 5.11 presents a comparative study of success rate and the number of function evaluations for all successful runs for the CMA-ES, DE, FDS algorithms, on 50 runs, max 2500 iterations, 80 chromosomes, for 2-dimensional functions.

Table 5. 11 Comparison of CMA-ES, DE and FDS algorithms in terms of mean number of function evaluations and success rate (50 runs, max 2500 iterations, 80 chromosomes).

function name	CMA-ES success rate	Function evaluations of CMA-ES	DE success rate	Function evaluations of DE	FDS success rate	Function evaluations of FDS
Easom	70%	17053	100%	3240	100%	12460
Matyas	100%	500	100%	2700	100%	840
Beale	100%	460	100%	3060	100%	1120
Booth's	100%	492	100%	2820	100%	1680
Goldstein–Price	100%	1812	100%	1620	100%	4900
Schaffer N.2	90%	6726	100%	5016	100%	2240
Schwefel's	0%	----	0%	----	**100%**	5460
Branins's rcos	100%	6876	100%	840	100%	1540
Six-hump camel	100%	780	100%	2160	100%	980
Shubert	90%	2220	100%	8160	100%	6300
Martin and Gaddy	100%	1660	100%	2400	100%	700
Michalewicz	100%	1848	0%	---	100%	7700
Drop-wave	50%	26470	94%	9048	100%	6300
Levy N. 13	100%	606	100%	1958	100%	2660
Rastrigin's	80%	13134	100%	2388	100%	8120
Sphere	100%	720	100%	1800	100%	980
Ackley d=4	100%	2240	100%	3480	86%	85760
Rosenbrock's	100%	1644	100%	4560	100%	2520

Figure 5. 8 shows that, for one of the tested functions (Schaffer), the solution has been found in 2 iterations by using the FDS algorithm.



Number of Dimensions n: 2

Function f: f = @(x) 0.5 + ((sin(x(:,1).^2-x(:,2).^2)).^2 - 0.5)./(1 + 0.001*(x(:,1).

Range Minimum a: [-100, -100]

Range Maximum b: [100, 100]

Number of Chromosomes M: 100

Optimum Fin: 0

Maximum Iterations: 1000

Optimization Task: Minimize

Optimum F: 0.00010614

	1	2
xval	0.2267	0.2214
mbits	21	21

Run Time: 0.017701 s

Number of Iterations: 2

Figure 5. 8 shows the solution of Schaffer N.2 function found in 2 iterations

5.4 The Multi Free Dynamic Schema (MFDS)

In this section we describe the Multi Free Dynamic Schema (MFDS) algorithm, which contains 5 types of operators (dynamic dissimilarity, similarity, dissimilarity, dynamic schema, free dynamic schema) and random generation of chromosomes. The free dynamic schema operator is applied 6 times. The dissimilarity, similarity, dynamic schema, dynamic dissimilarity operators and random generation were applied in the DDS algorithm, see Tables 3.1, 3.2, 4.2 and 4.3 for more details. The free dynamic schema operator was applied in FDS algorithm, see Table 5.8.

After noticing that the FDS algorithm was more effective than DSC, DSDSC and DDS algorithms in terms of speed in finding the best solution (a comparison of all algorithm is presented in Chapter 7), we now propose here another way of using the same principle as in FDS, but now a larger number of schema types (six) are selected at random from the first quarter of the sorted generation.

Suppose f is a one-dimensional function with domain [0, 1], as shown in Figure 5.9, This function is represented by binary representation consisting of four bits, (0000,0001,…,1111), that means the range is divided into 16 segments.

The principle of multi free schema is the following: Suppose there are K best solutions, and the free dynamic schema operator is applied when $R_1 = 3, R_2 = 2,\ldots, R_k = 3$. The same idea of free schema is used but here with more free schemas as shown in Figure 5.9. Here we discover multi free schema (101*), (0***), and so on. Then we randomly put 0 or 1 in positions having *s. Here the multi free schema will cover all the subspaces colored with red, as shown in Figure 5.9.

Bits:	1234
Ch.1:	1010
Schema :	101*
Sol.1:	1011
Sol.2:	1010

Here another example, the first bit is fixed.

Bits:	1234
Ch.1:	0100
Schema :	0***
Sol.1:	0000
Sol.2:	0001
Sol.3:	0010
….	
Sol.8:	0111

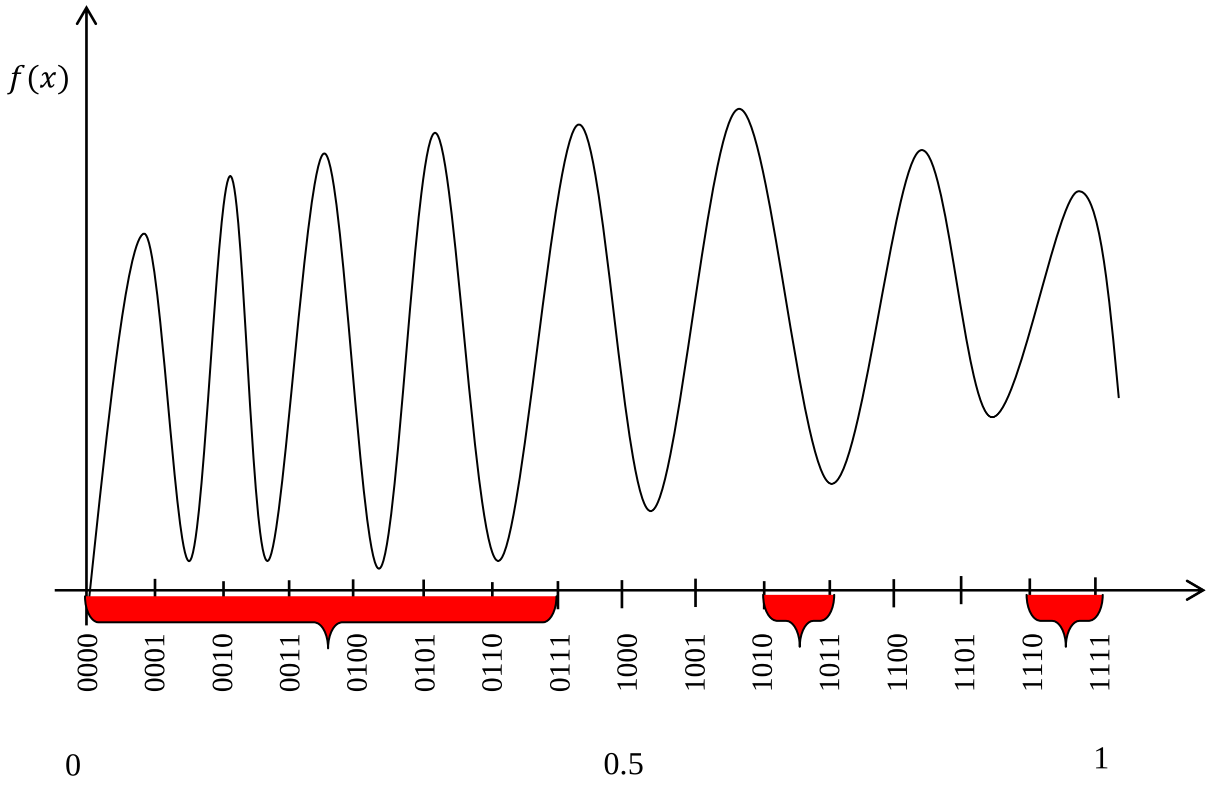


Figure 5. 9 Multi free dynamic schema.

5.4.1 Methodology

The MFDS algorithm starts with a random population (P0) of M elements representing a number of solutions to the problem. This population is sorted, then a new population (P1) is formed whose first 40% of chromosomes are copied from a part of (P0). The population (P0) is divided into for equal groups (G1, G2, G3, G4), then population (P1) is divides into 8 not equal groups (G5,G6,…,G12). Then we apply different operators to these groups (see Table 5. 12).

To groups (G1, G2, G3) of population (P0), the dynamic dissimilarity, similarity, dynamic schema operators are applied, and in (G4), random chromosomes are generated, respectively. To groups (G5, G6) of population (P1), the dissimilarity and dynamic dissimilarity operators are applied respectively, where each of (G5,G6) represents 20% of population (P1), For the next groups (G7,G8,…,G12), where each group represents 10% of population (P1), six types of free dynamic schema are applied, where chromosomes were randomly chosen from the first quarter of sorted population (P0), see Table 5. 12.

The free dynamic schema was mentioned in Table 5.8, but it was only used for the best chromosome Ch_1 in the population of the FDS algorithm. In this algorithm (MFDS) it is used six times, for different chromosomes and various random fixed part sizes R_i. Each free dynamic schema represents a group of solutions, these solutions are close to the area of best solutions because they are chosen form the first quarter in the sorted population (P0).

Table 5. 12 Populations (P0) and (P1) and the twelve groups of chromosomes.

ORIGINAL GROUPS OF CHROMOSOMES(P0)		**COPY GROUPS OF CHROMOSOMES(P1)**	
Ch_1	G1: To the first group the dynamic dissimilarity operator is applied.	**20% of population**	G5: To the fifth group the dissimilarity operator is applied.
Ch_2			
Ch. …			
Ch. …			
$Ch_{M/4}$		**20% of population**	G6: To the six group the dynamic dissimilarity operator is applied.
$Ch_{M/4+1}$	G2: To the second group the similarity operator is applied.		
$Ch_{M/4+2}$			
Ch. …			
Ch. …		**10% of population**	G7: To this group the free dynamic schema operator is applied.
$Ch_{M/2}$			
$Ch_{M/2+1}$	G3: To the third group the dynamic schema operator is applied.	**10% of population**	G8: To this group the free dynamic schema operator is applied.
$Ch_{M/2+2}$			
Ch. …		**10% of population**	G9: To this group the free dynamic schema operator is applied
Ch. …			
$Ch_{M/2+M/4}$		**10% of population**	G10: To this group the free dynamic schema operator is applied
$Ch_{M/2+M/4+1}$	G4: The fourth group is generated randomly.		
$Ch_{M/2+M/4+2}$		**10% of population**	G11: To this group the free dynamic schema operator is applied
Ch. …			
Ch. …		**10% of population**	G12: To this group the free dynamic schema operator is applied
Ch_M			

5.4.2 The MFDS algorithm

The following optimization problem is considered:

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}$$

$$\text{minimize}|\text{maximize } f(x_1, \dots, x_n) \text{ subject to}$$

$$x_i \in [a_i, b_i], i = 1, \dots, n$$

where $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is a given function.

In the algorithm described below, the encoding of chromosomes is the same as in Chapter 3.

Let *M* be a positive integer divisible by 8. The MFDS algorithm consists of the following steps:

1. Generate 2*M* chromosomes, each chromosome representing a point $(x_1, \dots, x_n)$. Divide the chromosomes into two populations (P0) and (P1), where (P0) consists of four groups (G1, G2, G3, G4), and (P1) consists of eight groups (G5, G6,…, G12), each group in (P0) having *M*/4 chromosomes, but in (P1) the size is equal to 20% of population for (G5, G6) and 10% for (G7, ... , G12) .
2. Compute the values of the fitness function f for each chromosome in the population (G1,…, G12).
3. Sort the chromosomes according to the descending (for maximization) or ascending (for minimization) values of the fitness function.
4. Copy the first 40% from (P0) onto (G5, G6), replacing the original chromosomes.
5. Copy *C* times the first chromosome and put it in *C* randomly chosen positions in the first half of population (P0), replacing the original chromosomes, where *C* = *M*/8.
6. Apply the dynamic schema operator for chromosomes A = Ch_1 and B = $Ch_{M/4}$ from populations (P0), (that is, the chromosomes on positions 1 and *M*/4, respectively). Copy this schema *M*/4 times and put it in (G3).
7. Apply the dynamic dissimilarity and similarity operators to groups (G1) and (G2) respectively. Apply the dissimilarity and dynamic dissimilarity operators to group (G5) and (G6) respectively.

8. Apply the free dynamic schema operator 6 times to generate six groups (G7,…, G12). To generate each group, a chromosome is chosen randomly from the first quarter of solutions in (P0). Then put 0 or 1 randomly in positions having *s in each group.
9. All the chromosomes created in Steps 4 to 8 replace the original ones in positions from 2 to 2M in populations (P0) and (P1). Then randomly generate chromosomes for group (G4).
10. Go to Step 2 and repeat until the stopping criterion is reached.

Note:

The stopping criterion for the algorithm depends on the example being considered, see Section 5.4.3. The free dynamic schema operator is shown in Table 5.8. that uses different sizes of fixed segments (gray color) and changing all the rest of chromosome by using (*)'s, then randomly put (0,1) to generate a new solutions.

The flowchart of MFDS algorithm is shown in Figure 5.10.

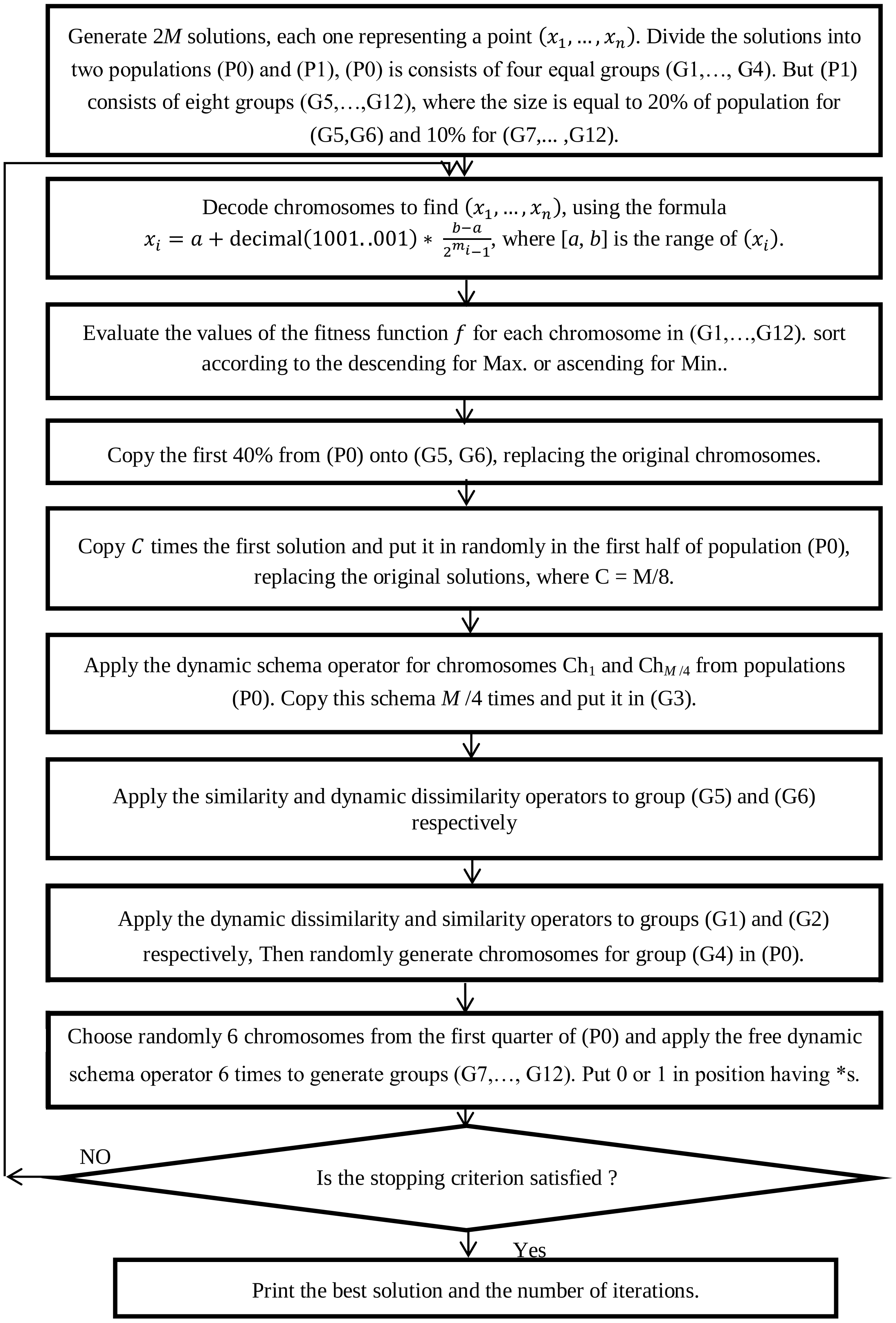

Generate 2M solutions, each one representing a point $(x_1, \ldots, x_n)$. Divide the solutions into two populations (P0) and (P1), (P0) is consists of four equal groups (G1,…, G4). But (P1) consists of eight groups (G5,…,G12), where the size is equal to 20% of population for (G5,G6) and 10% for (G7,... ,G12).
Decode chromosomes to find $(x_1, \ldots, x_n)$, using the formula $x_i = a + \text{decimal}(1001..001) * \frac{b-a}{2^{m_i-1}}$, where [$a$, b] is the range of (x_i).
Evaluate the values of the fitness function f for each chromosome in (G1,…,G12). sort according to the descending for Max. or ascending for Min..
Copy the first 40% from (P0) onto (G5, G6), replacing the original chromosomes.
Copy C times the first solution and put it in randomly in the first half of population (P0), replacing the original solutions, where C = M/8.
Apply the dynamic schema operator for chromosomes Ch_1 and $Ch_{M/4}$ from populations (P0). Copy this schema M /4 times and put it in (G3).
Apply the similarity and dynamic dissimilarity operators to group (G5) and (G6) respectively
Apply the dynamic dissimilarity and similarity operators to groups (G1) and (G2) respectively, Then randomly generate chromosomes for group (G4) in (P0).
Choose randomly 6 chromosomes from the first quarter of (P0) and apply the free dynamic schema operator 6 times to generate groups (G7,…, G12). Put 0 or 1 in position having *s.
NO
Is the stopping criterion satisfied ?
Yes
Print the best solution and the number of iterations.

Figure 5. 10 Flowchart of the MFDS algorithm.

5.4.3 Experimental results

In this section, we report on computational testing of the MFDS algorithm on 18 functions of 2 variables, one function of 4 variable, 5 functions of 10 variables and 100 variables, also the execution time is reported. After each test, the result of MFDS has been compared with the known global optimum and with the result of a CGA taken from taken from our experimental result (see Table 3.10), also, in Table 5.14 a comparison of the mean number of function evaluations and success rate of CMA-ES, DE and FDS algorithms is presented. All 22 tested functions with optimal solutions are mentioned in Appendix A. We have applied the algorithm with 80 chromosomes (P0) with the stopping criterion that the difference between our best solution and the known optimal solution is less than or equal a given threshold.

The MFDS algorithm has found optimum solutions for some optimization problems (like Beale's, Schaffer n.2, Schwefel's,) that the classical genetic algorithm cannot reach to 100% success rate with bit string or double vector, as shown in Table 5.13 the results for 50 runs of MFDS algorithm (the results for 50 runs of the MFDS algorithm). For our algorithm all success rates are 100% with 80 chromosomes in (P0) for all problems.

Table 5.14 presents a comparative study of success rate and the number of function evaluations for all successful runs for the CMA-ES, DE, MFDS algorithms, on 50 runs, max 2500 iterations, 80 chromosomes. Table 5.15. presents the results of MFDS for some 4- and 10-dimensional functions.

Figures 5.11, 5.12 present the average number of iterations with standard deviation of iterations for 2-, 4- and 10-dimensional functions for MFDS.

The MFDS algorithm keeps the best solution from each iteration at the first position until it is replaced by a better one.

Here it is possible to note the effect of multi free dynamic schema by decreasing the average number of iterations for most functions comparing with previous algorithms.

Table 5. 13 The results for 50 runs of the MFDS algorithm

Function name	Threshold	Min number of iterations/ Min time in seconds	Max number of iterations/ Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Success rate of MFDS	Rate of success GA
Easom	0.001	6	175	58	42.5	-0.99938	100%	100 % DV
		0.01869	0.2521	0.08722				
Matyas	0.001	2	9	5	1.5	0.000331	100%	100 % DV
		0.00660	0.01758	0.01101				
Beale's	0.001	2	24	7	3.7	0.000544	100%	70% DV
		0.00668	0.03775	0.0141				
Booth's	0.001	3	27	10	4.5	0.000545	100%	100 % DV
		0.00850	0.04361	0.01816				
Goldstein –Price	0.001	6	52	21	10.4	3.000434	100%	100 % DV
		0.0108	0.07839	0.03482				
Schaffer N.2	0.001	3	30	11	6.3	0.000402	100%	70% DV
		0.00661	0.04746	0.01967				
Schwefel's	0.001	4	69	29	14.1	0.000662	100%	0% BS
		0.0066	0.10088	0.04650				
Branins's rcos	0.001	2	60	7	8.1	0.398406	100%	100 % DV
		0.0066	0.08871	0.01480				
Six-hump camel back	0.001	2	15	5	2.7	-1.03119	100%	100 % DV
		0.006	0.02500	0.01208				
Shubert	0.01	2	75	26	15.2	-186.714	100%	100 % DV
		0.0149	0.12685	0.05036				
Martin and Gaddy	0.001	2	9	5	2	0.00043	100%	40% DV
		0.0067	0.01672	0.01121				
Michalewicz	0.04	4	116	30	24	38.8096	100%	80% DV
		0.0089	0.1614	0.0450				
Holder table	0.001	2	28	12	5	-19.2081	100%	80% DV
		0.0078	0.0435	0.02048				

Drop-wave	0.001	5	69	30	16.6	-0.99949	100%	100 % BS
		0.0165	0.1005	0.0467				
Levy N. 13	0.001	5	32	11	6	0.00045	100%	100 % BS
		0.0115	0.0496	0.02047				
Rastrigin's	0.001	7	85	41	19.7	0.000459	100%	100% BS
		0.0160	0.1202	0.05336				
Sphere	0.001	2	10	5	2	0.000452	100%	100% BS
		0.0128	0.0774	0.0202				
Rosenbrock's valley	0.001	2	40	13	9.7	0.000609	100%	100% BS
		0.0127	0.07139	0.0265				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

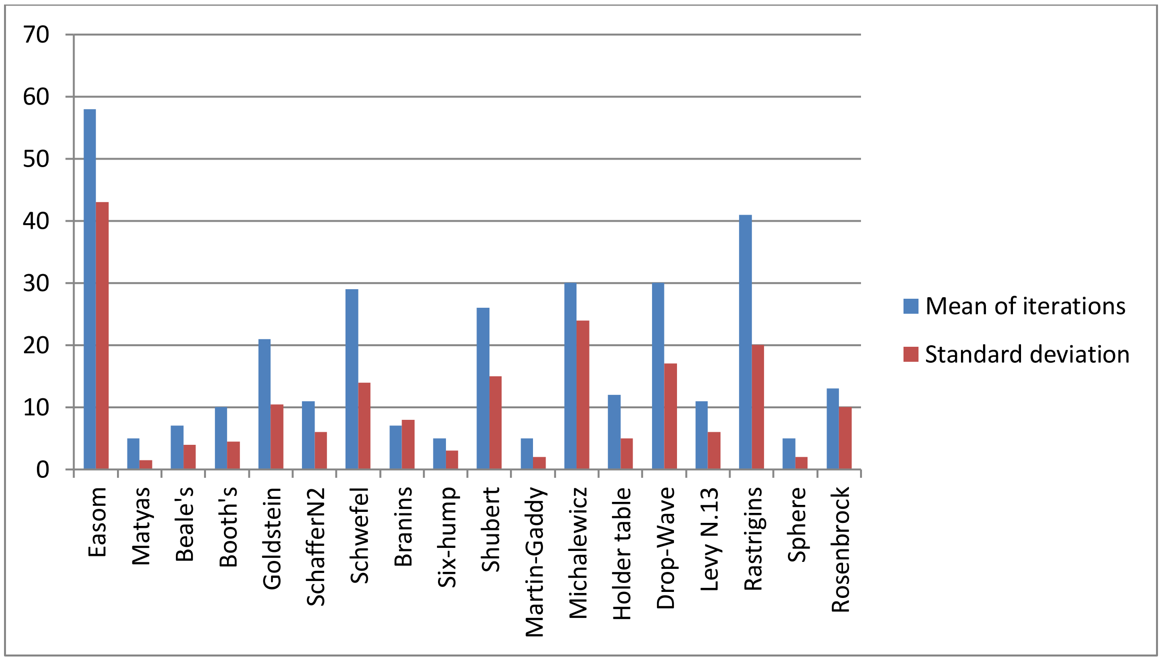


Figure 5. 11 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for MFDS algorithm

Table 5. 14 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and MFDS algorithms (50 runs, max 2500 iterations, 80 chromosomes)

function name	CMA-ES success rate	Function evaluations of CMA-ES	DE success rate	Function evaluations of DE	MFDS success rate	Function evaluations of MFDS
Easom	70%	17053	100%	3240	100%	8120
Matyas	100%	500	100%	2700	100%	700
Beale	100%	460	100%	3060	100%	980
Booth's	100%	492	100%	2820	100%	1400
Goldstein–Price	100%	1812	100%	1620	100%	2940
Schaffer N.2	90%	6726	100%	5016	100%	1540
Schwefel's	0%	**----**	0%	**----**	**100%**	4060
Branins's rcos	100%	6876	100%	840	100%	980
Six-hump camel	100%	780	100%	2160	100%	700
Shubert	90%	2220	100%	8160	100%	3640
Martin and Gaddy	100%	1660	100%	2400	100%	700
Michalewicz	100%	1848	0%	**---**	100%	4200
Drop-wave	50%	26470	94%	9048	100%	4200
Levy N. 13	100%	606	100%	1958	100%	1540
Rastrigin's	80%	13134	100%	2388	100%	5740
Sphere	100%	720	100%	1800	100%	700
Ackley d=4	100%	2240	100%	3480	100%	25760
Rosenbrock's	100%	1644	100%	4560	100%	1820

Table 5. 15 The results for 25 runs of the MFDS algorithm with execution time of 10-dimensional function.

Function name	Threshold	Min number of iterations/ Min time in seconds	Max number of iterations/ Max time in seconds	Mean no. of iterations for all successful runs	Std. of Iter.	Mean of the best solution fitness from all successful runs	Rate of success MFDS	Rate of success GA
Sum Squares d=10	0.01	77	438	245	100.1	0.07856	100%	100% BS
		0.18143	1.01937	0.58881				
Sphere d=10	0.01	19	68	46	10.8	0.08259	100%	100% BS
		0.04683	0.15147	0.10561				
Sum of Different Powers d=10	0.01	2	7	3	1.2	0.04680	100%	100% BS
		0.00375	0.02471	0.01004				
Zakharov d=10	0.1	107	700	373	156.6	0.09010	54%	100% BS
		0.32029	2.00436	1.07996				
Rastrigin d=10	0.01	157	456	294	70.8	0.07169	100%*	100% BS
		0.56427	1.64128	1.04986				
Ackley d=4	0.001	53	269	161	59.8	0.00071	100%*	100% BS
		0.07750	0.38244	0.22120				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

* For this function we change the R_i in dynamic schema and free dynamic schema to be a random number from $\{0, 1, \ldots, m_i\}$.

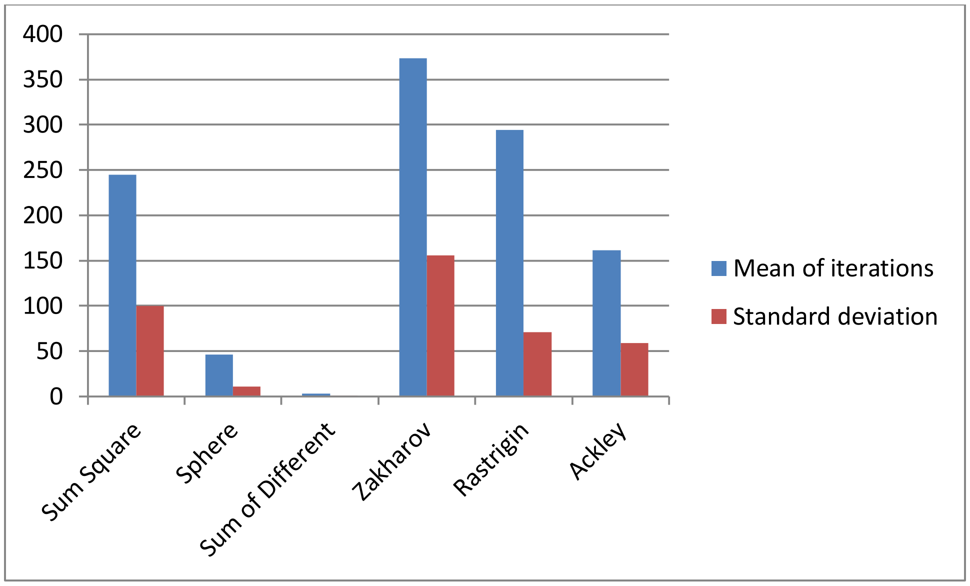


Figure 5. 12 The average number and standard deviation of iterations for 4- and 10-dimensional functions with 80 chromosomes for MFDS algorithm

The following Figures (5.13 and 5.14) show the behavior of a schema while finding the best solution for the Michalewicz function. It is clear that the chromosomes are focused on the best solution. Fig. 5.13 shows three dimensional view, the green points represent the population (P1), the blue points represent the population (P0). Fig. 5.14 shows the top view where the red points for population (P1) are focused on the best solution.

In Fig. 5.13, we can see different groups of green points which belong to different schemas, each schema has a group of solutions close together.

This algorithm can help searching or exploring different solutions in search space thus providing possibility in finding a solution with a lower number of iterations.

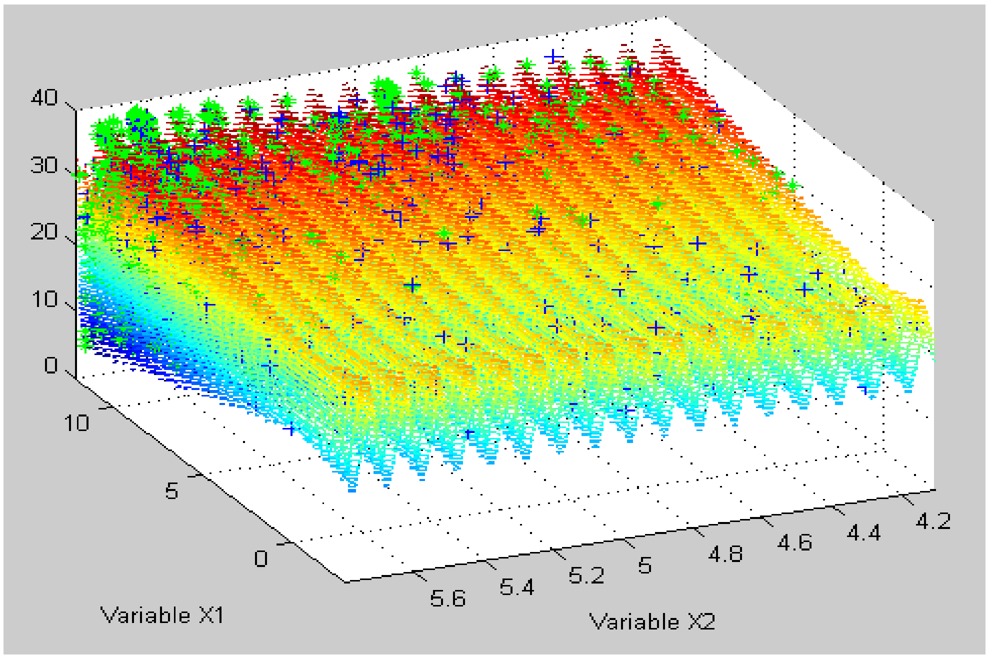


Figure 5. 13 shows the multi free dynamic schema on Michalewicz function finding best solution after 15 iterations (green points belong to the schema).

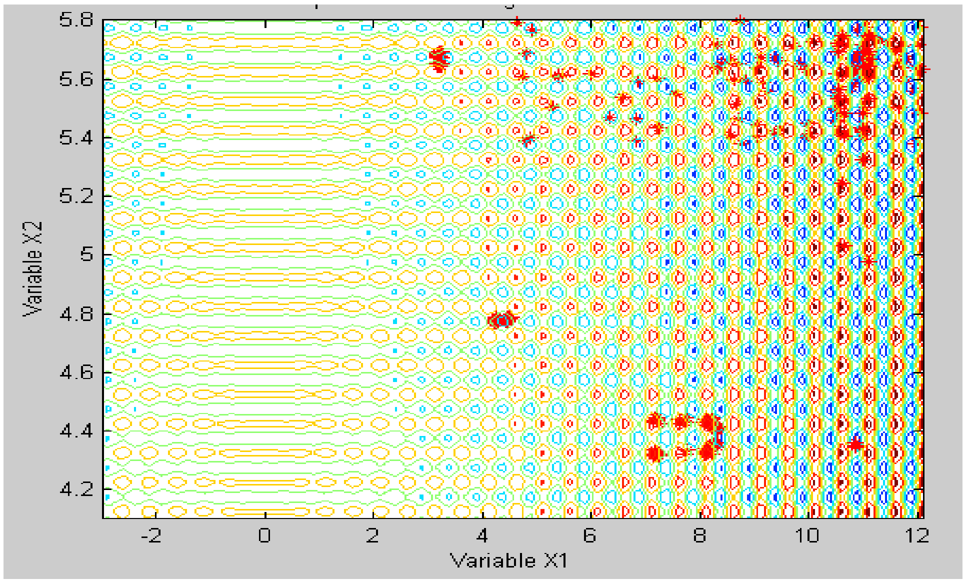
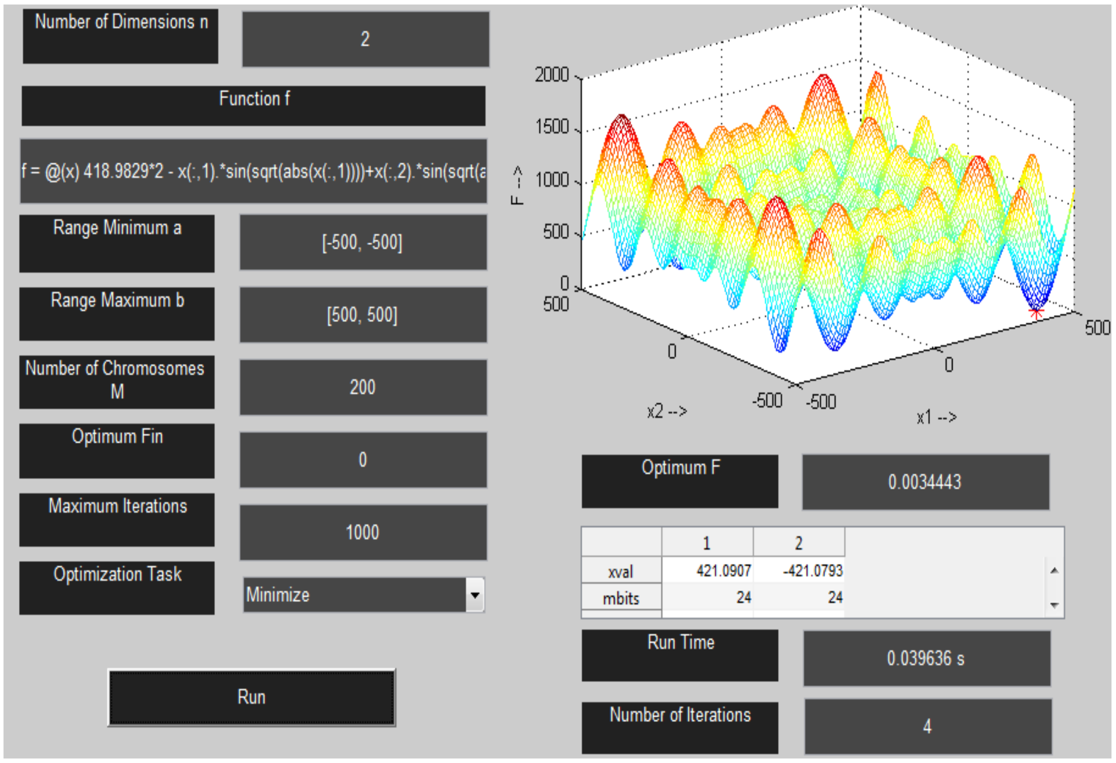


Figure 5. 14 shows the multi free dynamic schema on Michalewicz function finding the best solution (top view, where red points belong to the schema).

Fig. 5.15 shows one of the tested function (Schwefel's) where the MFDS algorithm found the optimum solution in 4 iterations.


Number of Dimensions n: 2

Function f: f = @(x) 418.9829*2 - x(:,1).*sin(sqrt(abs(x(:,1))))+x(:,2).*sin(sqrt(a

Range Minimum a: [-500, -500]

Range Maximum b: [500, 500]

Number of Chromosomes M: 200

Optimum Fin: 0

Maximum Iterations: 1000

Optimization Task: Minimize

Optimum F: 0.0034443

	1	2
xval	421.0907	-421.0793
mbits	24	24

Run Time: 0.039636 s

Number of Iterations: 4

Figure 5. 15 shows Schwefel's function, MFDS finding solution in 4 iterations

5.4.4 The choice of R_i for Rastrigin, Ackley and Zakharov functions

By changing the size of R_i parameter (for Rastrigin and Ackley functions) in range $\{0,1,\ldots,m_i\}$, we allow the gray part to contain all bits in x_i (we make a mask for the best or worst x_i randomly). This idea gives a chance to keep the best x_i without change and give a chance to the worst x_i to change all bits to be better. Each schema from 6 types of free dynamic schema could be like in the following Table 5. 16, in this example it's clear if $R_2 = 0$ and $R_3 = m_3$, this operation will keep x_3 without change and change all bits in x_2.

Table 5. 16 The size of R_i parameter on free dynamic schema $(0,\dots,m_i)$

No. of Ch.	m_1								m_2								m_3							
	R_1				$m_1 - R_1$				$m_2 - R_2$								R_3							
Ch_i	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1
$Schema_i$	0	0	0	0	*	*	*	*	*	*	*	*	*	*	*	*	0	0	0	0	0	0	1	1

On the other hand we faced a problem with the Zakharov function, by applying 10 dimensions with a range [-5, 10]. Since the result of success did not exceed 60%, we changed the parameter R_i to obtain 100% success rate for all dynamic operators (dynamic dissimilarity, dynamic schema, 6 types of free dynamic schema) as follows :

1. for dynamic dissimilarity we used $R_i = \{0,\dots,1/2m_i\}$.
2. for dynamic schema $R_i = \{0,1,\dots,m_i\}$. See Table 5.16.
3. for 6 types of free dynamic schema $R_i = 0$ or $R_i = m_i$. randomly for each $x = (x_i,\dots,x_n)$, this means we make a mask for $(x_i,\dots,x_n)$ randomly to keep the best x_i without change if $R_i = m_i$, or changing x_i by putting (*)s in all bits of x_i then generate {0,1} randomly instead of (*)s. See Table 5. 17.

In this example (Table 5. 17) the algorithm will keep the x_1 and x_3 without any change and change all bits of x_2.

Table 5. 17 The R_i parameter on free dynamic schema (0 or m_i).

No. of Ch.	m_1								m_2								m_3							
	$R_1 = m_1$								$R_2 = 0$								$R_3 = m_3$							
Ch_i	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1
$Schema_i$	0	0	0	0	0	0	1	0	*	*	*	*	*	*	*	*	0	0	0	0	0	0	1	1

Table 5.18 presents the results of 25 runs on 100-dimesional functions by using MFDS, CMA-ES and GA, we notice that CMA-ES have 0% success rate on Rastrigin function, in GA we have found the best solutions by using bit string and double vector with 200 chromosomes and two point crossover. We used 200 chromosomes, maximum 2000 iterations and threshold 0.1 for all algorithms in this comparison.

Table 5. 18 Comparing the success rate and mean number of iteration for 25 runs of the MFDS, GA, CMA-ES algorithms on 100-dimensional functions

Function name	Min number of iterations	Max number of iterations	Mean no. of iterations for all successful runs	Mean of the best solution fitness from all successful runs	Std.Dev. of mean no. of Iter.	Success rate of MFDS	Success rate of GA / Avr. of Iter.	Success rate of CMA-ES / Avr. of Iter.
Sum Squares d=100	479	1211	695	0.09675	119	100% ***	100% 273 It. DV	100% 541 It.
	7.33375	16.064	10.2286					
Sphere d=100	421	656	445	0.08958	47.5	100% ***	100% 234 It. DV	100% 286 It.
	5.3693	8.7836	6.5059					
Sum of Different Powers d=100	3	17	7	0.09685	3.3	100% ***	100% 85 It. DV	100% 89 It.
	0.0642	0.3157	0.1413					
Rastrigin d=100	505	1141	656	0.09864	127	100% ***	100% 218 It. DV	0% don't find
	6.6197	15.221	10.1945					
Ackley d=100	236	495	339	0.08955	87	100% ***	100% BS 97 It. 0% DV	100% 401 It.
	4.0450	9.1167	6.1949					

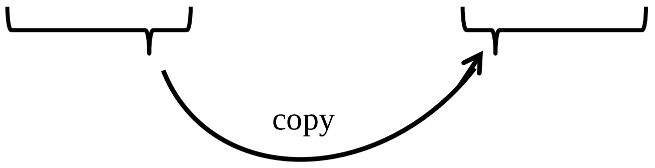
BS= bit string, DV= double vector as a parameter of population type in GA toolbox, with maximum 2000 iterations, two point crossover, 200 chromosomes, Std.Dev. = standard deviation.

*** For these functions we used copying the gray part from x_i to x_j as explained in Table 5.19.

For 100-dimensional functions, we have applied a new change in the MFDS algorithm: we appended one condition that if the number of dimensions is greater than 10, we apply coping the grey part of x_i to x_j in the best chromosome in a free dynamic schema procedure, where *i*≠*j*, *i*, *j* are chosen randomly as shown in Table 5.19 (the remaining bits in x_j don't change), this condition was added only for two groups of free dynamic schema in population (P1). Here we assume that R_i=R_j because we must copy the same number of highest bits from x_i to x_j.

Table 5. 19 The R_i parameter on free dynamic schema (0 or m_i).

	m_1	m_i						....	m_j						m_n
Ch_1	01010…010	0	0	0	0	1	0101	…….	0	0	0	0	1	010	01010…10100



CHAPTER SIX: Initial Population with Multi-Free Dynamic Schema

6.1 Introduction

To start solving an optimization problem by using a GA, the initial population generation is important. The population size usually remains the same in all generations. The main difficulty concerning the initial population is that randomly generated chromosomes may not satisfy the constraints of the problem. Another big difficulty is that the initial population can be disproportionate to the problem [103]. In [103] the population size was 100 chromosomes, but other authors used more than 100 in problems with very large solution spaces [104].

In [105] the authors proposed using a random size of the initial population. The minimum population size should be determined according to the problem size. The initial population is produced by randomly determining *p* chromosomes, where *p* is a population size [105].

In [106], [107] the influence of the population size is be discussed based on the Genetic Algorithm (GA) facility. The population is examined for the population with a fixed generations number, the examination is carried out between 5 and 200 chromosomes. For 200 generations, the optimal population size has been found to be 100 chromosomes [106], [107].

The population may contain non-useful and useful individuals. The operation is reasonably better if the population contains only useful individuals [108]. That is, the GA can faster obtain the best solution when the best chromosomes exist in the initial population.

A brief outline for a variety methods of maintaining population diversity was provided in [109]. The population diversity is effectively used to study the premature convergence. The degree of population diversity is directly associated with premature convergence [109], [110]. The mechanisms of preserving the diversity can help the optimization process in two ways. A diverse population is appropriate for dealing with multimodal functions and it can be used to simultaneously explore several landscape fitness hills. Diversity-preserving methods are able to support global exploration and able to help locating several local and global optima [110].

For the initial population, the essential challenge is that the individuals may not satisfy the restrictions of the problem. For reaching the optimal solution by some successive generation, the GA must improve the populations' individuals [111].

The chromosomes are randomly generated, in order to collect the initial population. The population size 100 chromosomes was considered in [112] and this number of chromosomes has been utilized in previous studies such as [113], [104], also in examples with big solution spaces, the initial population should be larger than 100.

Numerical experiments highlighted that the populations utilizing very small or very large number of chromosomes number could lead to attain insignificant solutions [114]. In addition, the population size of a range between 20 to 60 was applied and employed to three problem types in [115].

The population size can be considered as one of the most important topics of the evolutionary computation. An argument is commonly raised that a "small" population size can guide the algorithm to poor solutions and a "large" population size can lead to make the algorithm to spend more computation time until finding a solution [116]. The center of mass was suggested to be used as an alternative method for measuring the diversity of the population level. This theoretical approach of the initial random population diversity analysis and measure is important. Also, it could be necessary for designing the GAs because of the initial population relations with other GA parameters and because of its relations to the premature convergence problem [116].

In [117], the researchers studied the effect of the first generation as well as the diversity of the first generation on reaching the optimal solution of GAs. There is a hypothesis saying that "higher diversity in initial populations for Genetic Algorithms can reduce the number of iterations required to reach an optimum and potentially increase solution quality" [117]. It seems that for small populations it may be better to generate structured chromosomes than random ones, and diversity can help to measure how structured the initial population is [117].

In this chapter a new optimization algorithm (IPMFDS) is proposed taking advantage of the effect of initial population. We have used a big initial population of 500, 1000 and 3000 chromosomes for 2-, 10- and 100-dimensional functions, respectively. This initial population is evaluated and sorted, then we choose a number of the best chromosomes to form first population in the IPMFDS algorithm. This first population is of size 80, 160 and 200 chromosomes for 2-, 10- and 100-dimensional functions, respectively. By this method, when the best chromosomes are present in the initial population, the algorithm can find the optimal solution very fast.

6.2 The IPMFDS algorithm

The following optimization problem is considered:

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}$$
$$\text{minimize}|\text{maximize } f(x_1, \dots, x_n) \text{ subject to}$$
$$x_i \in [a_i, b_i], i = 1, \dots, n$$

where $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is a given function.

In the algorithm described below, the encoding of chromosomes is the same as in Chapter 3.

Let *M* be a positive integer divisible by 8. The IPMFDS algorithm consists of the following steps:

1. Generate a big initial population (P), with size corresponding to the number of variables in the problem (i.e., 500, 1000, 3000 for 2, 10, 100 variables respectively), of chromosomes which represent the points $(x_1, \dots, x_n)$. Then decode, evaluate and sort fitness function values of chromosomes in ascending order for minimization, and descending order for maximization, then get the best $2M$ chromosomes that can be collected from the initial population.
2. Put the *2M* chromosomes in the population (P0) and (P1), where (P0) consists of four groups (G1, G2, G3, G4), and (P1) consists of eight groups (G5, G6,…, G12),

each group in (P0) having *M*/4 chromosomes, but in (P1) the size is equal to 20% of population for (G5, G6) and 10% for (G7, ... , G12).

3. Compute the values of the fitness function *f* for each chromosome in the population (G1,…, G12).
4. Sort the chromosomes according to the descending (for maximization) or ascending (for minimization) values of the fitness function.
5. Copy the first 40% from (P0) onto (G5, G6), replacing the original chromosomes.
6. Copy *C* times the first chromosome and put it in *C* randomly chosen positions in the first half of population (P0), replacing the original chromosomes, where $C = M/10$.
7. Apply the dynamic schema operator for chromosomes $A = Ch_1$ and $B = Ch_{M/4}$ from populations (P0), (that is, the chromosomes on positions 1 and *M*/4, respectively). Copy this schema *M*/4 times and put it in (G3).
8. Apply the dynamic dissimilarity and similarity operators to groups (G1) and (G2) respectively. Apply the dissimilarity and dynamic dissimilarity operators to group (G5) and (G6) respectively.
9. Apply the free dynamic schema operator 6 times to generate six groups (G7,…, G12), to generate each free schema chromosome is chosen randomly from first quarter of solution in (P0). The put 0 or 1 randomly in positions having *s in each group.
10. All the chromosomes created in Steps 5 to 9 replace the original ones in positions from 2 to 2*M* in populations (P0) and (P1). Then randomly generate chromosomes for group (G4).
11. Go to Step 3 and repeat until the stopping criterion is reached.

Note: The stopping criterion for the algorithm depends on the example being considered, see Section 5.4.3.

The flowchart of IPMFDS algorithm is shown in Figure 6. 1.

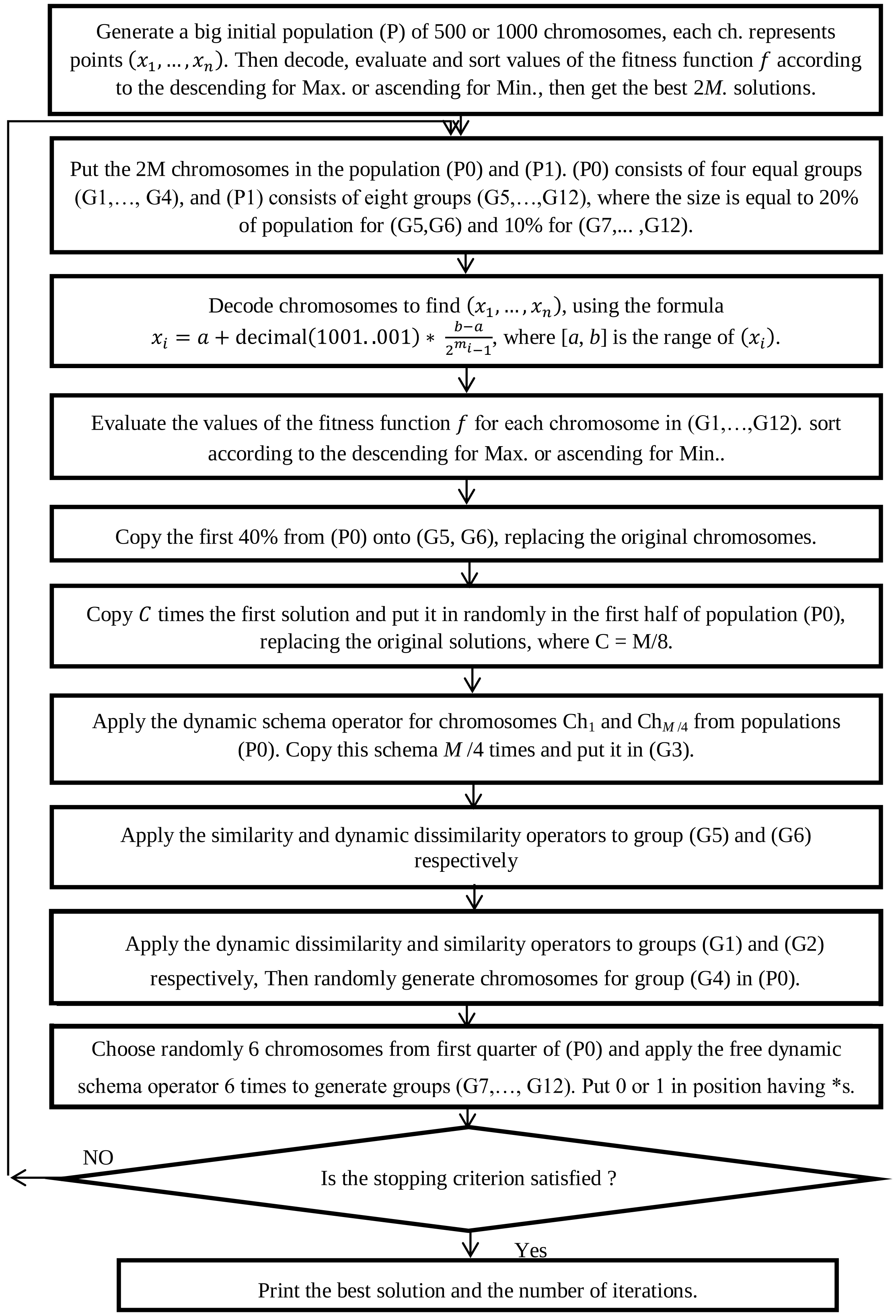

Generate a big initial population (P) of 500 or 1000 chromosomes, each ch. represents points $(x_1, \dots, x_n)$. Then decode, evaluate and sort values of the fitness function f according to the descending for Max. or ascending for Min., then get the best $2M$. solutions.
Put the 2M chromosomes in the population (P0) and (P1). (P0) consists of four equal groups (G1,…, G4), and (P1) consists of eight groups (G5,…,G12), where the size is equal to 20% of population for (G5,G6) and 10% for (G7,... ,G12).
Decode chromosomes to find $(x_1, \dots, x_n)$, using the formula $x_i = a + \text{decimal}(1001..001) * \frac{b-a}{2^{m_i-1}}$, where $[a, b]$ is the range of (x_i).
Evaluate the values of the fitness function f for each chromosome in (G1,…,G12). sort according to the descending for Max. or ascending for Min..
Copy the first 40% from (P0) onto (G5, G6), replacing the original chromosomes.
Copy C times the first solution and put it in randomly in the first half of population (P0), replacing the original solutions, where C = M/8.
Apply the dynamic schema operator for chromosomes Ch_1 and $Ch_{M/4}$ from populations (P0). Copy this schema $M/4$ times and put it in (G3).
Apply the similarity and dynamic dissimilarity operators to group (G5) and (G6) respectively
Apply the dynamic dissimilarity and similarity operators to groups (G1) and (G2) respectively, Then randomly generate chromosomes for group (G4) in (P0).
Choose randomly 6 chromosomes from first quarter of (P0) and apply the free dynamic schema operator 6 times to generate groups (G7,…, G12). Put 0 or 1 in position having *s.
NO
Is the stopping criterion satisfied ?
Yes
Print the best solution and the number of iterations.

Figure 6. 1 The flowchart of the IPMDS algorithm.

6.3 Experimental results

In this section, we report on computational testing of the IPMFDS algorithm on 18 functions of 2 variables, one function of 4 variable, 5 functions of 10 variables and 5 functions of 100 variables , also the execution time is reported. After each test, the result of IPMFDS has been compared with the known global optimum and with the result of a CGA taken from our experimental results . All 22 tested functions with optimal solutions are mentioned in Appendix A. We have applied the algorithm with the initial population (P) as given in the Introduction: 500 chromosomes when applied to 2 dimensions, and set (P0) to 80 chromosomes, see Table 6.1. But the initial population (P) of 1000, 3000 chromosomes is used with 10-, 100-dimensional functions, see Tables 6.1, 6.3, 6.4. The stopping criterion is that the difference between our best solution and the known optimal solution is less than or equal to a given threshold, see Tables 6.1, 6.3, 6.4.

Table 6.2 presents a comparative study of success rate and the number of function evaluations for all successful runs for the CMA-ES, DE, IPMFDS algorithms, on 50 runs, max 2500 iterations, 80 chromosomes.

Figure 6. 2, 6.3 present the average number of iterations with standard deviation of iterations for 2-dimensional and 10-dimensional functions for IPMFDS algorithm.

The IPMFDS algorithm has found optimum solutions for some optimization problems (like Beale's, Schaffer N.2, Schwefel's,) that the classical genetic algorithm cannot reach to 100% success rate with bit string or double vector for population type, as shown in Table 6.1, column nine. For our algorithm all success rates are 100% with 80 chromosomes in (P0) for all problems.

Note that the average number of iterations was really low to find the optimal solutions, this was by effect of the big initial population.

The IPMFDS algorithm keeps the best solution from each iteration at the first position until it is replaced by a better one.

Figure 6.4 shows the implementation of Shubert function with small range [-4, 4], here it’s clear the IPMFDS algorithm has found the optimum solution in one iteration after applying a big initial population on this small range.

Table 6. 1 The results for 50 runs of the IPMFDS algorithm (80 chromosomes in P0).

Function name	Threshold of stopping criteria	Min number of iterations/ Min time in seconds	Max number of iterations/ Max time in seconds	Mean no. of iterations for all successful runs/ Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Success rate of IPMFDS	Rate of success GA
Easom	0.001	5	235	38	46	-0.99938	100%	100% DV
		0.0140	0.3685	0.06620				
Matyas	0.001	2	7	4	1.2	0.000415	100%	100% DV
		0.00711	0.0161	0.01120				
Beale's	0.001	2	27	8	5	0.000403	100%	70% DV
		0.00688	0.0427	0.01581				
Booth's	0.001	3	29	8	4	0.000447	100%	100% DV
		0.0083	0.0442	0.01597				
Goldstein–Price	0.001	3	22	11	4.4	3.00057	100%	100% DV
		0.00872	0.0388	0.02115				
Schaffer N.2	0.001	2	20	8	3.6	0.000368	100%	70% DV
		0.00688	0.0337	0.01677				
Schwefel's	0.001	2	27	15	5.9	0.000484	100%	0% BS
		0.01295	0.0836	0.0277				
Branins's rcos	0.001	2	144	17	29	0.398312	100%	100% DV
		0.00680	0.2047	0.0287				
Six-hump camel back	0.001	2	60	8	10.8	-1.03115	100%	100% DV
		0.00659	0.0933	0.0158				
Shubert	0.01	2	39	11	6.8	-186.717	100%	100% DV
		0.0056	0.0629	0.0218				
Martin and Gaddy	0.001	2	8	4	1.4	0.000395	100%	40% DV
		0.00681	0.0154	0.0103				
Michalewicz	0.04	2	234	27	34.4	38.81845	100%	80% DV
		0.00887	0.3313	0.0464				

Holder Table	0.001	3	15	7	3	-19.2081	100%	80% DV
		0.00767	0.0273	0.0149				
Drop-wave	0.001	2	117	23	21	-0.9995	100%	100% BS
		0.01294	0.1679	0.0379				
Levy N. 13	0.001	2	30	10	5.4	0.000405	100%	100% BS
		0.00693	0.0479	0.0188				
Rastrigin's	0.001	7	66	33	13.1	0.000408	100%*	100% BS
		0.01007	0.0730	0.0427				
Sphere	0.001	2	14	6	3.3	0.000419	100%	100% BS
		0.00180	0.0285	0.0194				
Rosenbrock's valley	0.001	2	35	15	9.4	0.000559	100%	100% BS
		0.00429	0.0495	0.0268				

BS= bit string, DV= double vector as a parameter of population type in GA toolbox, Std.Dev. = standard deviation.

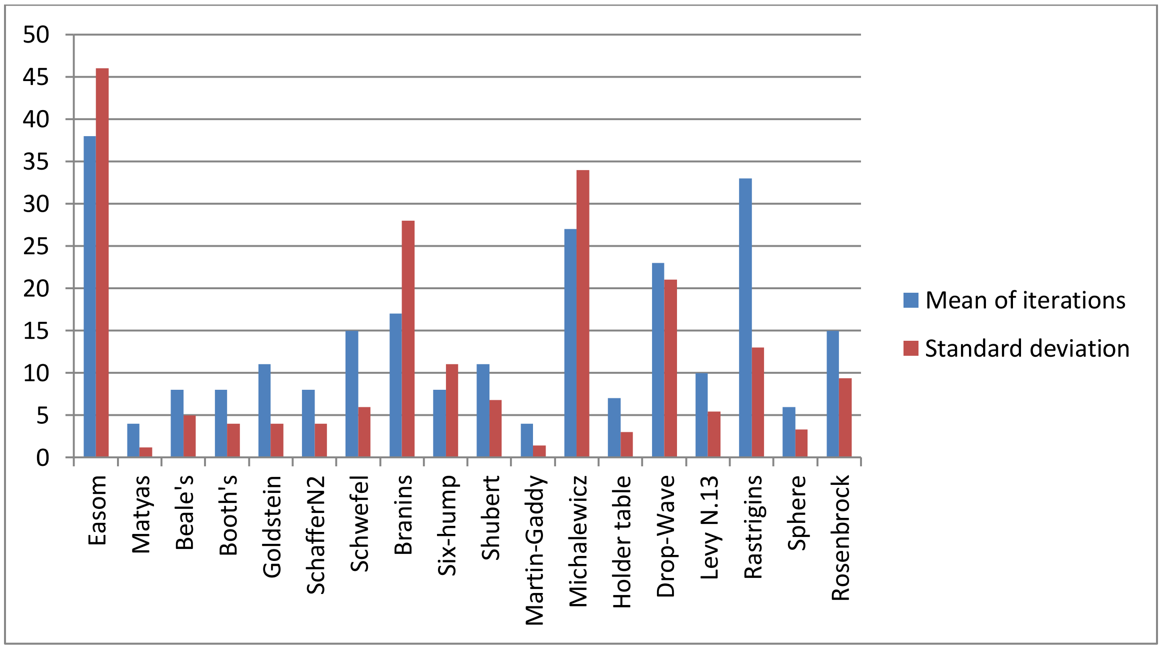


Figure 6. 2 The average number and standard deviation of iterations for 2-dimensional functions with 80 chromosomes for IPMFDS algorithm

Table 6. 2 Comparing the mean number of function evaluations and success rate of CMA-ES, DE and IPMFDS algorithms (50 runs, max 2500 iterations, 80 chromosomes)

function name	CMA-ES success rate	Function evaluations of CMA-ES	DE success rate	Function evaluations of DE	IPMFDS success rate	Function evaluations of IPMFDS
Easom	70%	17053	100%	3240	100%	6080
Matyas	100%	500	100%	2700	100%	640
Beale	100%	460	100%	3060	100%	1280
Booth's	100%	492	100%	2820	100%	1280
Goldstein–Price	100%	1812	100%	1620	100%	1760
Schaffer N.2	90%	6726	100%	5016	100%	1280
Schwefel's	0%	**----**	0%	**----**	**100%**	2400
Branins's rcos	100%	6876	100%	840	100%	2720
Six-hump camel	100%	780	100%	2160	100%	1280
Shubert	90%	2220	100%	8160	100%	1760
Martin and Gaddy	100%	1660	100%	2400	100%	640
Michalewicz	100%	1848	0%	---	100%	4320
Drop-wave	50%	26470	94%	9048	100%	3680
Levy N. 13	100%	606	100%	1958	100%	1600
Rastrigin's	80%	13134	100%	2388	100%	5280
Sphere	100%	720	100%	1800	100%	960
Ackley d=4	100%	2240	100%	3480	100%	8160
Rosenbrock's	100%	1644	100%	4560	100%	2400

Table 6. 3 The results for 50 runs of the IPMFDS algorithm and run time of 10-dimensional function.

Function name	Threshold	Min number of iterations / Min time in seconds	Max number of iterations / Max time in seconds	Mean no. of iterations for all successful runs / Average time	Std.Dev. of mean no. of Iter.	Mean of the best solution fitness from all successful runs	Success rate of IPMFDS	Success rate of GA
Sum Squares d=10	0.1	127	388	268	76.9	0.076726	100%	100% BS
		0.396452	1.171682	0.79964				
Sphere d=10	0.1	112	602	351	145.6	0.016648	100%	100% BS
		0.263907	1.383436	0.81086				
Sum of Different Powers d=10	0.1	2	9	4	2	0.012048	100%	100% BS
		0.007757	0.033951	0.019697				
Zakharov d=10	0.1	337	2000	1209	631	1.561844	60%	100% BS
		0.975018	5.721531	3.465392				
Rastrigin d=10	0.01	129	482	277	195	0.081194	100%*	100% BS
		0.466429	1.722902	0.99826				
Ackley d=4	0.001	31	94	51	42	0.007941	100%*	100% BS
		0.055756	0.135876	0.080212				

* In this function we change the R_i in dynamic schema and free dynamic schema to be in the random range $\{0, \dots, m_i\}$ in the grey parts.

Table 6.4 presents the success rate, for CMA-ES, GA and IPMFDS, for 5 test functions of 100 variables, with min and max the number of iterations given for IPMFDS only. It is clear that the CMA-ES algorithm fails to find the solutions for Rastrigin function with 100-dimensions with threshold = 0.1, this table also shows the average number of iterations for CMA-ES and GA. In GA, we have found the best solutions by using bit string or double vector with 200 chromosomes and two point crossover. For

CMA-ES and IPMFDS we have also used 200 chromosomes, and maximum 2000 iterations for all algorithms in this comparison.

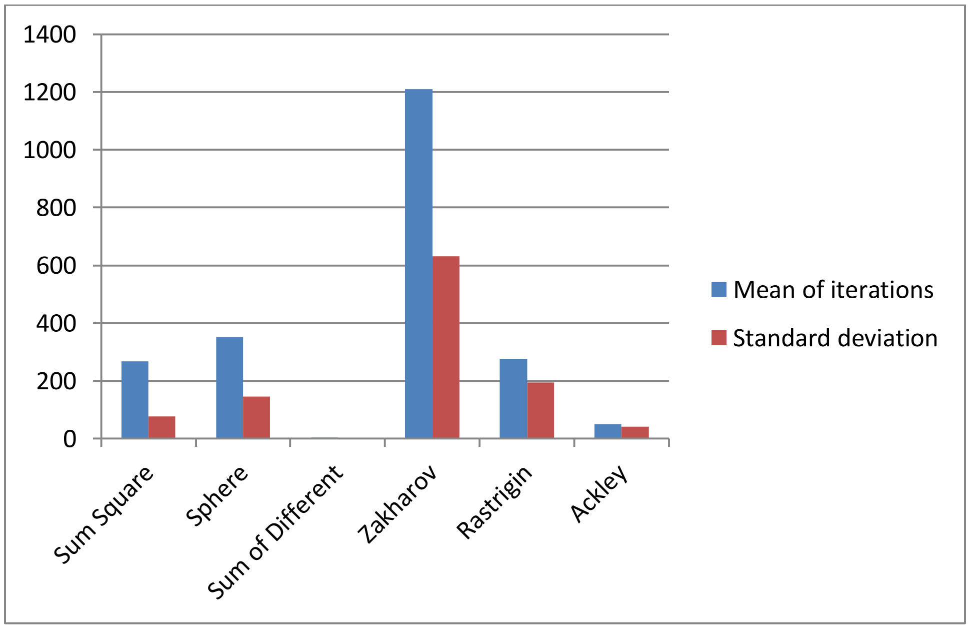


Figure 6. 3 The average number and standard deviation of iterations for 10-dimensional functions with 80 chromosomes for IPMFDS algorithm

Table 6. 4 Comparing the success rate and mean number of iterations on 25 runs of the IPMFDS, GA, CMA-ES algorithms of 100-dimensional functions

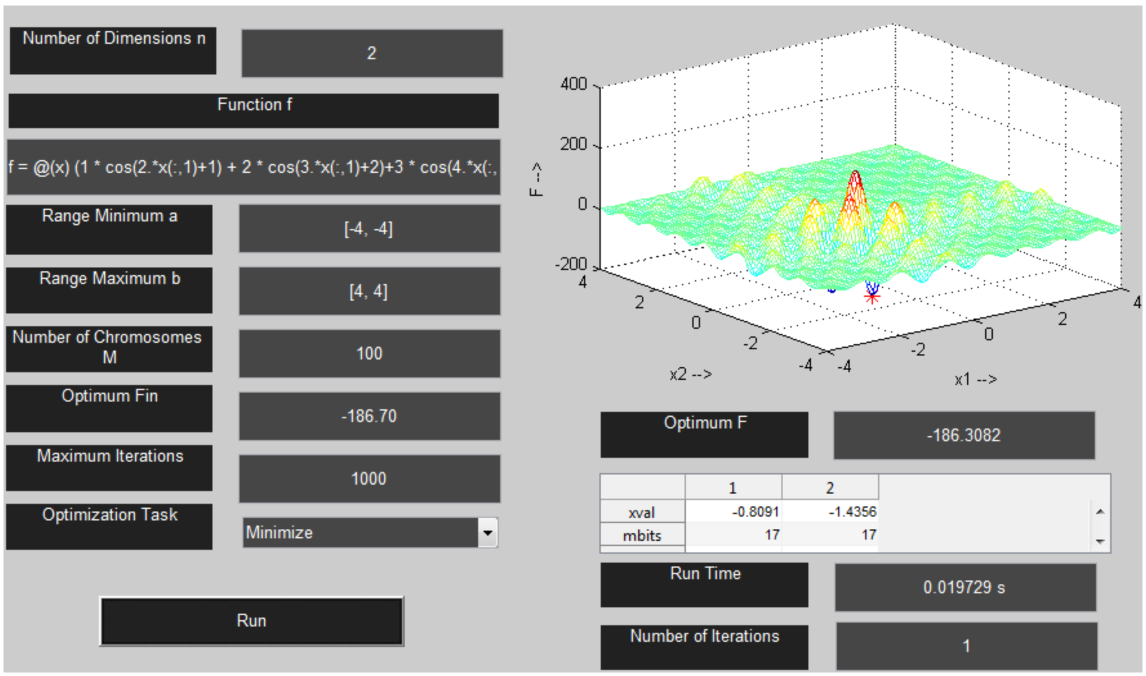
Function name	**Min number of iterations**	**Max number of iterations**	**Mean no. of iterations for all successful runs**	**Mean of the best solution fitness from all successful runs**	**Std.Dev. of mean no. of Iter.**	**Success rate of IPMFDS**	**Success rate of GA / Avr. of Iter.**	**Success rate of CMA-ES / Avr. of Iter.**
Sum Squares d=100	495	784	635	0.09899	108	100% ***	100% 273 It. DV	100% 541 It.
	8.8878	17.98339	12.50090					

Sphere d=100	359	612	486	0.08995	62	100% ***	100% 234 It. DV	100% 286 It.
	7.2347	11.32734	8.864059					
Sum of Different Powers d=100	3	16	8	0.083024	3.04	100% ***	100% 85 It. DV	100% 89 It.
	0.2160	0.50533	0.335193					
Rastrigind d=100	409	1130	643	0.09586	159	100% ***	100% 218 It. DV	0% Don’t find
	6.892259	19.76294	11.94528					
Ackely d=100	289	630	369	0.08465	84	100% ***	100% BS 97 It. 0% DV	100% 401 It.
	4.045013	10.82567	7.94879					

*** For all functions, we apply the condition that if the number of dimensions is greater than 10, we make copy of the high significant bits from x_i to x_j in the same way as described in Chapter 5, see Table 5. 19.

Fig. 6.4 shows how we can reach the optimum solution in 1 iteration by using a small area of search for the Shubert function. Figure 6.5 shows a top view of Michalewicz function to display the behavior of IPMFDS algorithm, the population (P1) is colored by red, it’s clear that it focuses on the area of best solution, on the other hand the blue points are for population (P0), that have more distribution in the search space for this function, this figure was made for 19 iterations with the first population of 500 chromosomes.

The Figure 6.6 shows the three-dimensional view of Michalewicz function, it shows how the best solution is handled. Solutions are concentrated near the optimal solution as shown in green points, on the other hand blue solutions represent the random re-generation of a part of the population, this figure was taken after 7 iterations. This means that the IPMFDS algorithm has the ability to search in the best area of a function.



Number of Dimensions n	2
Function f	f = @(x) (1 * cos(2.*x(:,1)+1) + 2 * cos(3.*x(:,1)+2)+3 * cos(4.*x(:,
Range Minimum a	[-4, -4]
Range Maximum b	[4, 4]
Number of Chromosomes M	100
Optimum Fin	-186.70
Maximum Iterations	1000
Optimization Task	Minimize
Optimum F	-186.3082
Run Time	0.019729 s
Number of Iterations	1

	1	2
xval	-0.8091	-1.4356
mbits	17	17

Figure 6. 4 Shubert function: one iteration with small range [-4,4].

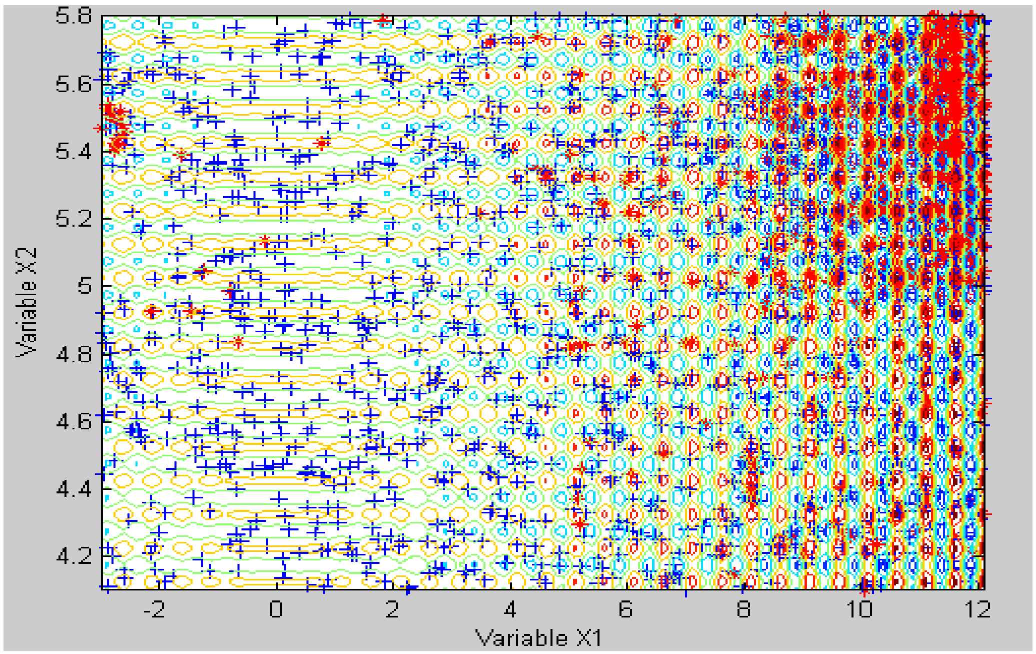


Figure 6. 5 The behavior of population (P0, P1) after 19 iterations for Michalewicz function.

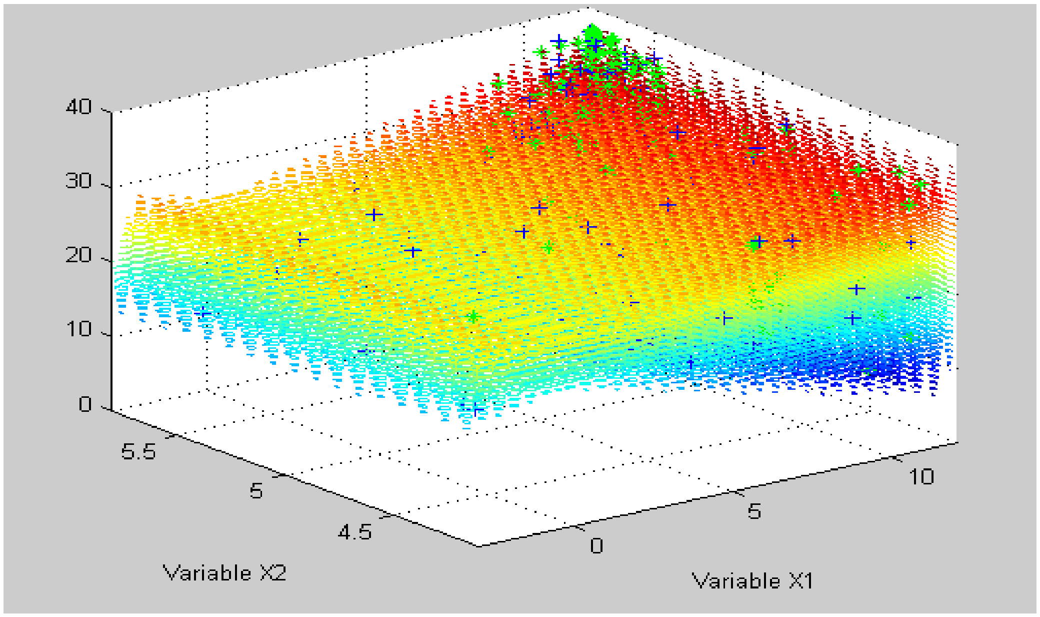


Figure 6. 6 The IPMFDS algorithm has the ability to search in the best area of Michalewicz function.

CHAPTER SEVEN: Comparison of all algorithms on selected continuous and combinatorial optimization problems

7.1 Comparison of all algorithms

In this section three types of comparison are reported, according to the following criteria: the average number of iterations, the average execution time, the average number of function evaluations and the success rate. This shows the performance of all algorithms introduced in this thesis: DSC, DSDSC, DDS, FDS, MFDS, IPMFDS, with figures that present graphically the obtained values.

7.1.1 Comparison of the average number of iterations

In this subsection we compare the average number of iterations for all six algorithms described in this thesis (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS), and for known algorithms (CMA-ES, DE, GA), for all test functions with 2 dimensions, as shown in Table 7.1. Figure 7.1 presents the values from Table 7.1, it's clear that MFDS and IPMFDS have the minimum average number of iterations to find the solutions for most tested functions.

Table 7.2 presents the comparison of the average number of iterations for 10-dimensional problems for all six algorithms from this thesis (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS), and Figure 7.2 shows the values from Table7.2.

Table 7. 1 Comparing the average number of iterations for 2-dimensional functions for all algorithms.

Function name	DSC	DSDSC	DDS	FDS	MFDS	IPMDFS	CMA-ES	DE	GA DV
Easom	88	51	62	89	58	38	384	41	124
Matyas	31	11	5	6	5	4	9	34	125
Beale's	93	49	16	8	7	8	8	38	204
Booth's	151	20	17	12	10	8	8	35	75
Goldstein–Price	134	34	20	35	21	11	31	21	82
Schaffer N.2	278	71	14	16	11	8	112	63	93
Schwefel's	561	41	65	39	29	15	*	*	*
Branins's rcos	86	28	9	11	7	17	115	11	68
Six-hump camel back	39	18	8	7	5	8	13	27	75
Shubert	198	19	33	45	26	11	37	102	64
Martin and Gaddy	36	15	6	5	5	4	28	30	320
Michalewicz	207	67	71	55	30	27	31	500	72
Holder Table	47	12	24	19	12	7	*	113	240
Drop-wave	201	48	44	45	30	23	441	25	*
Levy N. 13	290	45	19	19	11	10	10	20	*
Rastrigin's	71	25	38	58	41	33	219	23	51
Sphere	75	7	4	7	5	6	12	44	63
Ackley d=4	348	644	805	536	161	51	38	57	*
Rosenbrock's	101	115	24	18	13	15	27	41	*

*Doesn't find solution.

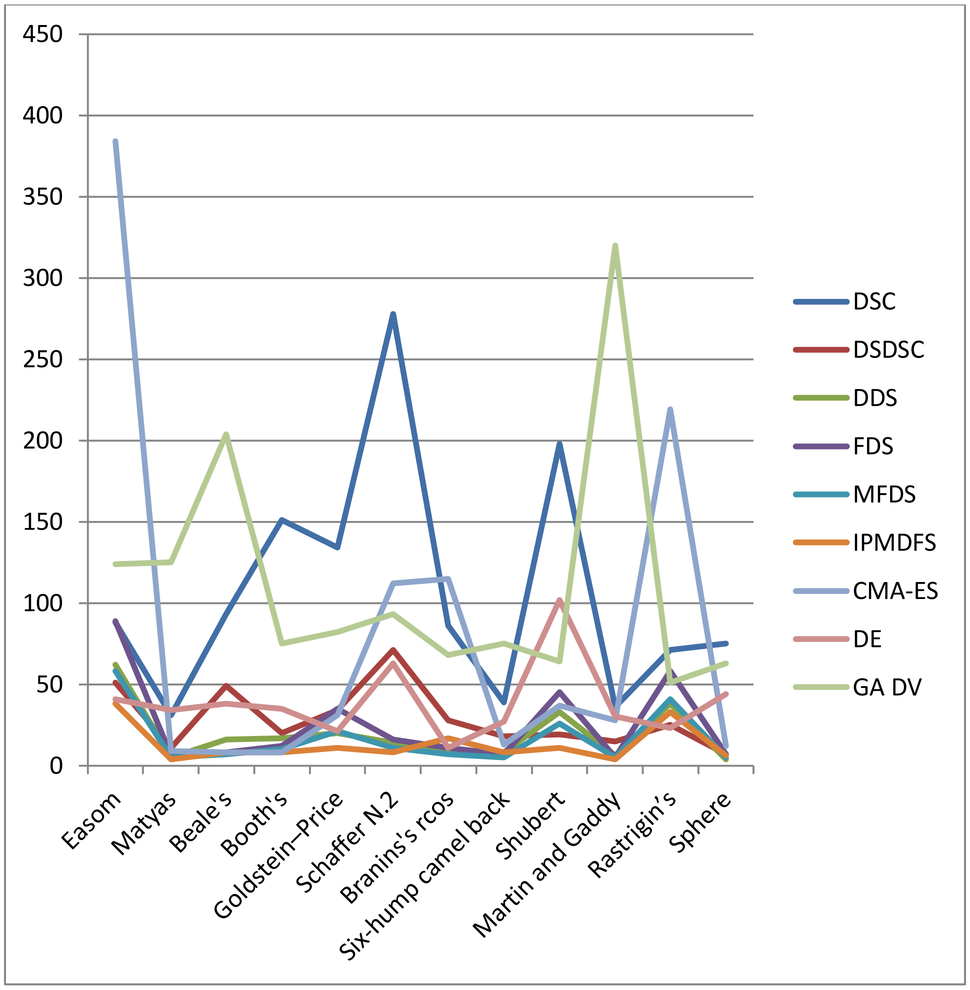


Figure 7. 1 Comparing the average number of iterations for 2-dimensional functions for all algorithms

Table 7. 2 Comparing the average number of iterations for 10-dimensional functions with 160 chromosomes for all algorithms

Function name	**DSC**	**DSDSC**	**DDS**	**FDS**	**MFDS**	**IPMDFS**
Sum Squares d=10	1936	145	128	320	245	268
Sphere d=10	746	31	23	51	46	351

Sum of Different Powers d=10	14	3	4	4	3	4
Zakharov d=10	1808	217	468	581	373	1209
Rastrigin d=10	***	1045	***	1159	294	277

*** No success for this function.

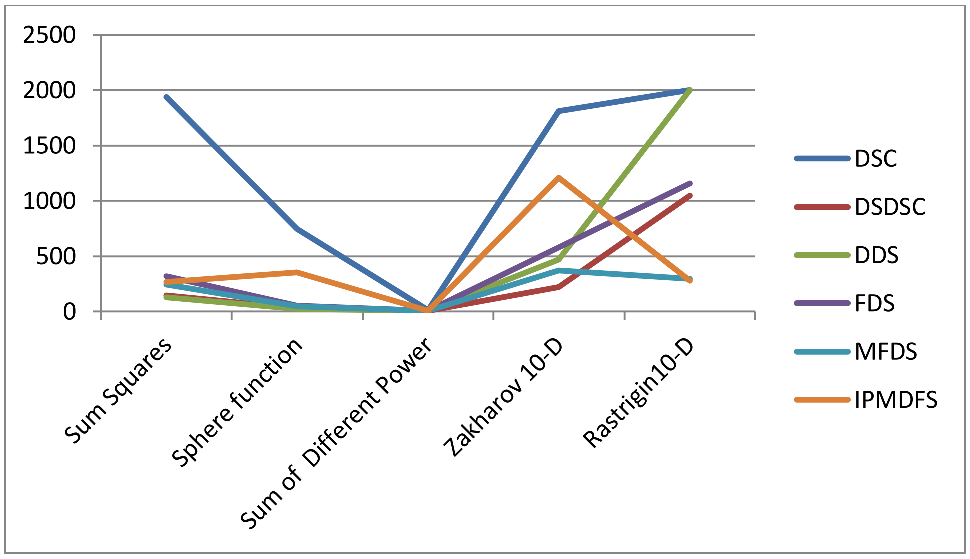


Figure 7. 2 Comparing the average number of iterations for 10-dimensional functions for all algorithms.

7.1.2 Comparison of the average run time

In this subsection we present a comparison of the average run time among all six algorithms that are described in this thesis (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS) for all tested functions with 2 dimensions, as shown in Table 7.3. Figure 7.3 presents the values from Table7.3, it is clear that the IPMFDS algorithm is the faster one, for most functions . These results are obtained on a computer with 2.4 MHz core i5, 8 GB RAM.

Table 7. 3 Comparing the average run time for 2-dimensional functions for all algorithms.

Function name	DSC	DSDSC	DDS	FDS	MFDS	IPMDFS
Easom	0.057025	0.05162	0.06854	0.10246	0.08722	0.06620
Matyas	0.022821	0.01257	0.00890	0.01018	0.01101	0.01120
Beale's	0.059205	0.05682	0.02033	0.01230	0.01412	0.01581
Booth's	0.104111	0.02052	0.02088	0.01525	0.01816	0.01597
Goldstein–Price	0.048197	0.03643	0.02487	0.04092	0.03482	0.02115
Schaffer N.2	0.094244	0.07470	0.01865	0.02046	0.01967	0.01677
Schwefel's	0.560628	0.04776	0.07268	0.04629	0.04650	0.02770
Branins's rcos	0.045151	0.02526	0.0136	0.01717	0.01480	0.02879
Six-hump camel back	0.018381	0.02446	0.01255	0.01105	0.01208	0.01587
Shubert	0.114426	0.02082	0.04216	0.05621	0.05036	0.02188
Martin and Gaddy	0.015214	0.01696	0.00970	0.0093	0.01123	0.01035
Michalewicz	0.12288	0.03951	0.07212	0.06059	0.04506	0.04642
Holder Table	0.031067	0.01882	0.02977	0.02392	0.02042	0.01496
Drop-wave	0.090239	0.05398	0.04936	0.05029	0.04676	0.03797
Levy N. 13	0.195895	0.04869	0.02487	0.02350	0.02047	0.01888
Rastrigin's	0.04296	0.02036	0.04922	0.05550	0.05336	0.04279
sphere	0.03764	0.01222	0.01252	0.01612	0.02026	0.01942
Rosenbrock's valley	0.04330	0.05360	0.03077	0.02552	0.02657	0.02680

Figure 7. 3 presents the comparison of all algorithms by time for 2-dimensional functions, it is clear that the IPMFDS algorithm is the fastest one, for most functions .

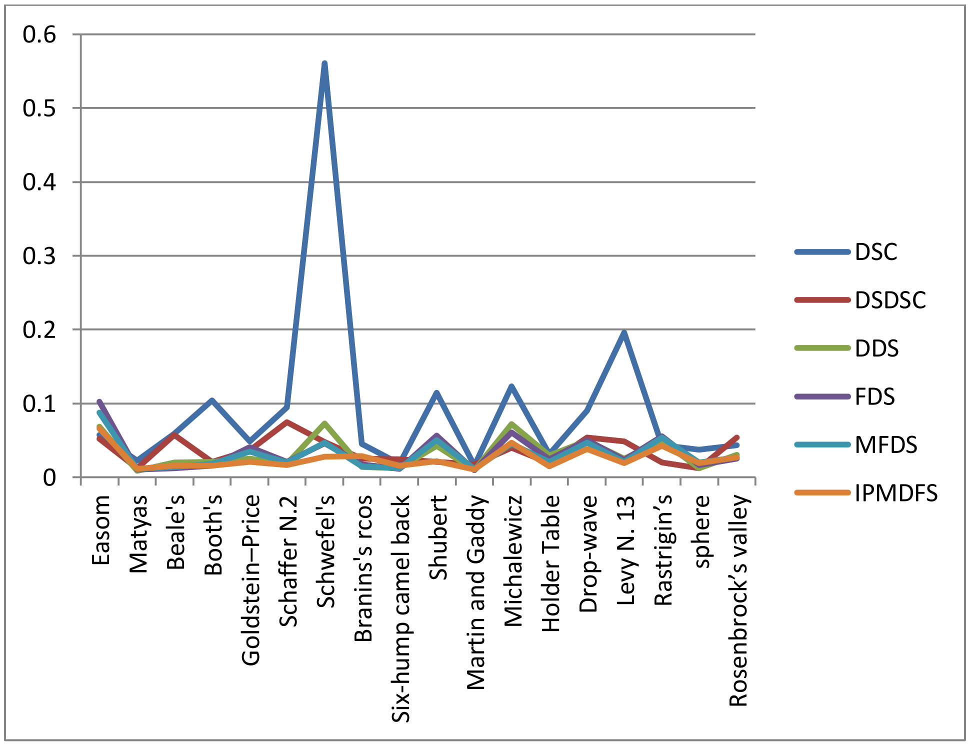


Figure 7. 3 Comparing the average run time for 2-dimensional functions for all algorithms.

Below, a comparison of the average run time is presented among all six algorithms that are described in this thesis (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS) for all tested functions with 10 dimensions, as shown in Table 7.4. Figure 7.4 presents the values from Table 7.4.

Table 7. 4 Comparing the average of run time for 10-dimensional functions for all algorithms.

Function name	DSC	DSDSC	DDS	FDS	MFDS	IPMDFS
Sum Squares d=10	2.872696	0.221053	0.335435	0.611585	0.588812	0.79964
Sphere d=10	1.055842	0.046462	0.05829	0.092892	0.105611	0.81086
Sum of Different Powers d=10	0.024399	0.005628	0.011352	0.011509	0.010049	0.019697
Zakharov d=10	2.847633	0.288333	1.649469	1.550803	1.079965	3.465392
Rastrigin d=10	2.668789	1.378663	4.388916	3.975838	1.049863	0.99826

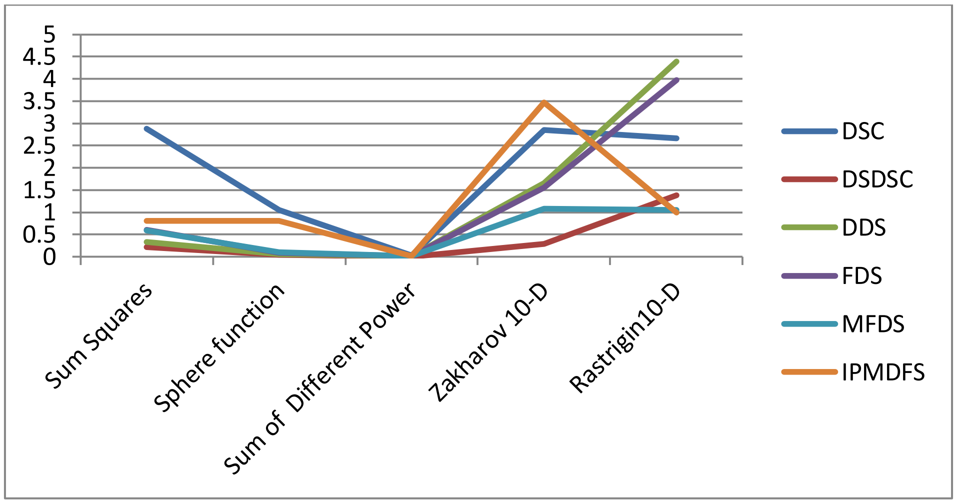


Figure 7. 4 Comparing the average run time for 10-dimensions functions for all algorithms.

It can be noted from the previous Table 7.4 that the run time rate was improved in most cases, especially when we used 1000 elements in the first initial generation for 10 dimensions. Also, as shown in Figure 7. 4, it is clear the fastest algorithms for most functions were DSDSC and MFDS.

7.1.3 Comparison of the number of function evaluations and the success rate

In this subsection a comparison of the average number of function evaluations is presented for all the six algorithms that are described in this thesis (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS) and for two known algorithms (CMA-ES, DE), for all the test functions of 2 variables. The results are shown in Table 7. 5 and presented on a diagram in Figure 7.5. Also, a comparison of the success rates is presented in Table 7.6.

Table 7. 5 Comparing the average number of function evaluations for 2-dimensional functions with CMA-ES and DE algorithms

Function name	DSC	DSDSC	DDS	FDS	MFDS	IPMDFS	CMA_ES	DE
Easom	7040	4080	8680	12460	8120	6080	17053	3240
Matyas	2480	880	700	840	700	640	500	2700
Beale's	7440	3920	2240	1120	980	1280	460	3060
Booth's	12080	1600	2380	1680	1400	1280	492	2820
Goldstein–Price	10720	2720	2800	4900	2940	1760	1812	1620
Schaffer N.2	22240	5680	1960	2240	1540	1280	6726	5016
Schwefel's	44880	3280	9100	5460	4060	2400	---	---
Branins's rcos	6880	2240	1260	1540	980	2720	6876	840
Six-hump camel	3120	1440	1120	980	700	1280	780	2160
Shubert	15840	1520	4620	6300	3640	1760	2220	8160
Martin and Gaddy	2880	1200	840	700	700	640	1660	2400
Michalewicz	16560	5360	9940	7700	4200	4320	1848	---
Holder Table	3760	960	3360	2660	1680	1120	---	---

Drop-wave	16080	3840	6160	6300	4200	3680	26470	9048
Levy N. 13	23200	3600	2660	2660	1540	1600	606	1958
Rastrigin's	5680	2000	5220	8120	5740	5280	13134	2388
Sphere	75	560	560	980	700	960	720	1800
Ackley d=4	30240	90160	112700	85760	25760	8160	2240	3480
Rosenbrock's	101	9200	3360	2520	1820	2400	1644	4560

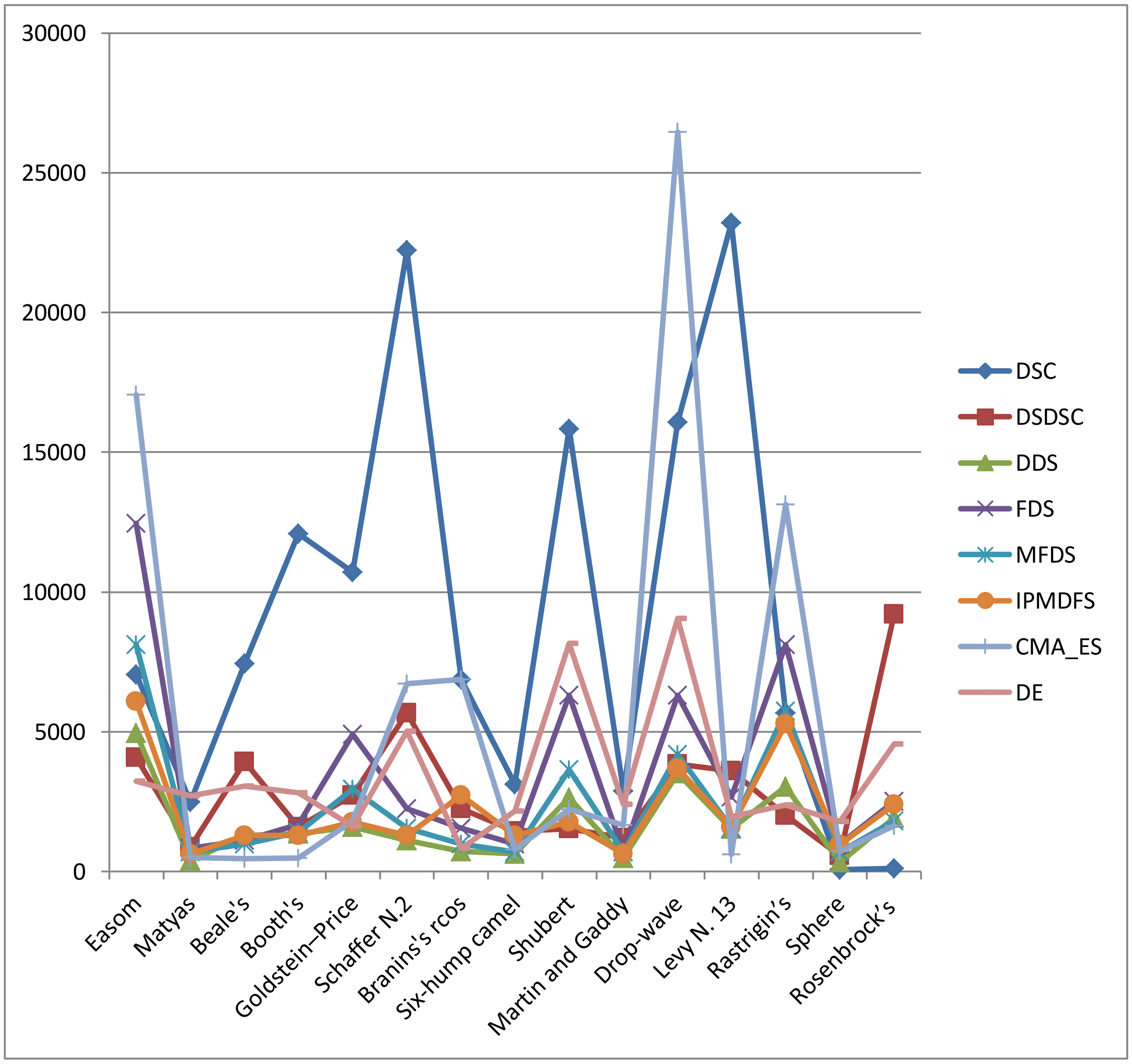


Figure 7. 5 Comparing the average number of function evaluations for 2-dimensional functions for all algorithms.

It seems obvious that the DDS and IPMDFS algorithms have the lowest numbers of function evaluations comparing with others algorithms, for most of the tested functions, see Figure 7.5.

Table 7.6 presents a comparison of success rate, for 9 algorithms (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS, CMA-ES, DE, GA), for 2-dimensional test functions, with population size 80 chromosomes and maximum 2500 iterations.

Table 7. 6 The success rate for 2-dimensional functions for all our algorithms comparing with CMA-ES, DE and GA with 80 chromosomes, max. 2500 iterations

Function name	**DSC**	**DSDSC**	**DDS**	**FDS**	**MFDS And IPMFDS**	**CMA_ES**	**DE**	**GA**
Easom	100%	100%	100%	100%	100%	70%	100%	100% DV
Matyas	100%	100%	100%	100%	100%	100%	100%	100% DV
Beale's	100%	100%	100%	100%	100%	100%	100%	70% DV
Booth's	100%	100%	100%	100%	100%	100%	100%	100% DV
Goldstein–Price	100%	100%	100%	100%	100%	100%	100%	100% DV
Schaffer N.2	100%	100%	100%	100%	100%	90%	100%	70% DV
Schwefel's	**92%**	100%	100%	100%	100%	0%	0%	0% BS
Branins's rcos	100%	100%	100%	100%	100%	100%	100%	100% DV
Six-hump camel	100%	100%	100%	100%	100%	100%	100%	100% DV
Shubert	100%	100%	100%	100%	100%	90%	100%	100% DV
Martin and Gaddy	100%	100%	100%	100%	100%	100%	100%	40% DV
Michalewicz	100%	100%	100%	100%	100%	100%	0%	80% DV
Holder Table	100%	100%	100%	100%	100%	---	---	80% DV

Drop-wave	100%	100%	100%	100%	100%	50%	94%	100% BS
Levy N. 13	100%	100%	100%	100%	100%	100%	100%	100% BS
Rastrigin's	100%	100%	100%	100%	100%	80%	100%	100% BS
Sphere	100%	100%	100%	100%	100%	100%	100%	100% BS
Ackley d=4	100%	**80%**	**50%**	**86%**	100%	100%	100%	100% BS
Rosenbrock's	100%	100%	100%	100%	100%	100%	100%	100% BS

Below, a comparison of the number of function evaluations and success rate is presented, for four algorithms (CMA-ES, GA, MFDS, IPMFDS), for five tested functions of 100 variables, with threshold equal to 0.1. The results are shown in Table 7.7.

Table 7. 7 Comparing the number of function evaluation and the success rate for CMA-ES, GA, MFDS and IPMFDS algorithms for 25 runs on 100-dimensional functions.

Function name	**CMA-ES Mean of function evaluation**	**Success rate of CMA-ES**	**GA Mean of function evaluation**	**Success rate GA**	**MFDS Mean of function evaluation**	**Success rate of MFDS**	**IPMFDS Mean of function evaluation**	**Success rate of IPMFDS**
Sum Squares d=100	81324	100%	54600	100% DV	139000	100%	127635	100%
Sphere d=100	51636	100%	48200	100% DV	89000	100%	97200	100%
Sum of Different Powers d=100	16164	100%	17000	100% DV	1600	100%	1600	100%
Rastrigin d=100	Don't find solution	0%	43600	100% DV	128600	100%	128600	100%
Ackley d=100	72180	100%	19400	100% BS	67800	100%	73800	100%

7.2 Application of all algorithms on some functions from the CEC 2017 benchmark (2- and 3-dimensional shifted and rotated functions)

In this section, we report on computational testing of 8 algorithms (DSC, DSDSC, DDS, FDS, MFDS, IPMFDS, CMA-ES, DE), on five two-dimensional shifted and rotated functions (Bent Cigar, Sum of Different Power, Zakharov, Rosenbrock's, Rastrigin's) [45], by using 300 chromosomes, maximum 5000 iterations and 100 runs. Table 7.8 presents these results.

We have also performed an experiment on two three-dimensional shifted and rotated functions (Bent Cigar, Sum of Different Power). We have applied three algorithms (IPMFDS, CMA-ES, DE), by using 300 chromosomes, maximum 5000 iterations and 100 runs. We notice that for the Shifted and Rotated Bent Cigar function the CMA-ES algorithm has not found the optimum solution, the IPMFDS has found the solution only with 2% success rate, while the DE algorithm is the best one that has found optimum solution with 100% success rate. On the other hand, for Shifted and Rotated Sum of Different Power, all three algorithms have got 100% success rate and the IPMFDS is the fastest one. Table 7.9 presents these experimental results.

Table 7. 8 The results for 25 runs of all our algorithms comparing with CMA-ES and DE on 2-dimensional of shifted and rotated functions.

Function name	DSC			DSDSC			DSS			FDS		
	Success rate	Mean of iteration	Mean of function evaluations	Success rate	Mean of iteration	Mean of function evaluations	Success rate	Mean of iteration	Mean of function evaluations	Success rate	Mean of iteration	Mean of function evaluations
Shif. Rot. Bent Cigar	12%	802	64160	12%	1246	99680	20%	1520	121600	24%	1427	114160
Shif. Rot. Sum of Different Power	100%	240	19200	100%	21	1680	100%	25	2000	100%	29	2320
Shif. Rot. Zakharov	100%	394	31520	100%	22	1760	100%	31	2480	100%	45	3600
Shif. Rot. Rosenbrock's	32%	711	56880	72%	895	71600	96%	490	39200	96%	286	22880
Shif. Rot. Rastrigin's	50%	1197	95760	100%	408	32640	100%	363	29040	100%	318	25440

Shif. Rot. = Shifted and Rotated.

Function name	MFDS			IPMDSC			CMA-ES			DE		
	Success rate	**Mean of iteration**	**Mean of function evaluations**	**Success rate**	**Mean of iteration**	**Mean of function evaluations**	**Success rate**	**Mean of iteration**	**Mean of function evaluations**	**Success rate**	**Mean of iteration**	**Mean of function evaluations**
Shif. Rot. Bent Cigar	85%	1138	90240	92%	1031	82480	100%	24	1920	100%	119	9520
Shif. Rot. Sum of Different Power	100%	11	880	100%	14	1120	100%	12	960	100%	40	3200
Shif. Rot. Zakharov	100%	10	800	100%	15	1200	100%	14	1120	100%	41	3280
Shif. Rot. Rosenbrock's	100%	263	21040	100%	238	19040	100%	45	3600	100%	107	8560
Shif. Rot. Rastrigin's	100%	74	5920	100%	102	8160	0% (doesn't find the solution)			100%	66	5280

Shif. Rot. = Shifted and Rotated.

We used these values: $o = (50, 20)$, $M = \begin{bmatrix} \cos(30) & \sin(30) \\ \sin(30) & \cos(30) \end{bmatrix}$ for sifted and rotated functions.

Table 7. 9 Application of the IPMDSC, CMA-ES and DE algorithms on two 3-dimensional shifted and rotated functions from CEC 2017 with 100 runs.

Function name	**IPMDSC**			**CMA-ES**			**DE**		
	Success rate	**Mean of iteration**	**Mean of function evaluation**	**Success rate**	**Mean of iteration**	**Mean of function evaluation**	**Success rate**	**Mean of iteration**	**Mean of function evaluation**
Shif. Rot. Bent Cigar	2%	55	16500	0% (doesn't find the solution)			100%	316	94800
Shif. Rot. Sum of Different Power	100%	4	1200	100%	9	2700	100%	39	11700

In three-dimensional rotated functions we used these values:

$$o = (50, 20, 30), M = [R_x \quad R_y \quad R_z],$$

where $R_x, \quad R_y, \quad R_z$ are:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where θ is equal to 30.

7.3 Application of our algorithms to the knapsack problem

The knapsack problem or rucksack problem is a problem in combinatorial optimization: given a set of items, each with a weight and a value, determine the count of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most useful items [118], [119].

In [120] the authors studied a 0-1 knapsack problem. There are many problems in this category, NP-hard, and large cases of such problems can only be addressed using heuristic algorithms. They analyzed experimentally the behavior of a few GA-based algorithms on several sets of randomly generated test problems. They used in all experiments the population size equal to 100, mutation and crossover rates fixed to 0.05 and 0.65, respectively; the authors used a simple one-point crossover. As a performance measure, they used the best solution found within 500 generations.

The formulation of a knapsack problem can be utilized to describe other problems like, for example, the feature selection problem that frequently occurs in the context of construction of an analytical model [118], [121].

The most common problem being solved is the 0-1 knapsack problem, which restricts the number x_i of copies of each kind of item to zero or one. Given a set of n items numbered from 1 up to n, each with a weight w_i and a value v_i, along with a maximum weight capacity W,

$$\text{maximize } f(x) = \sum_{i=1}^{n} v_i .\ x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i .\ x_i \leq W \text{ and } x_i \in \{0,1\} \qquad (7.1)$$

Here $f(\cdot)$ represents the fitness function and $x = (x_1, \dots, x_n)$ is a binary vector, indicating selected items, i.e., $x_i = 1$ if the i-th item is selected into the knapsack, and $x_i = 0$ otherwise [118].

Informally, the problem is to maximize the sum of the values of the items in the knapsack so that the sum of the weights is less than or equal to the knapsack's capacity.

Example of problem (7.1) with 50 items:

We have taken the GA program and data of a knapsack problem from [122], where $W = 625$.

weight-set =
[23,47,22,15,42,30,15,32,47,33,15,38,44,7,16,34,30,33,3,2,43,31,46,17,30,1,34,21,30,21,29,21,36,14,18,21,13,3,27,44,33,11,9,31,40,40,30,9,41,31]

price-set =
[27,34,9,22,8,17,22,21,23,19,7,36,11,42,37,16,10,26,10,50,23,46,37,3,14,16,35,14,15,44,49,2,45,3,15,1,34,44,19,25,43,28,26,4,30,24,49,11,48,13];

In Table 7.10, we have applied 7 algorithms to the knapsack problem with 150 and 500 iterations, with population size 80. We notice that in 150 iterations the GA has not reached to the highest value for given data, but DSDSC has found the best value (915) as maximum value and IPMFDS is found the next best value (912) as maximum value. These algorithms have reached solutions better than GA (904) and other algorithms. On the other hand, we notice that in 500 iterations the GA, DSC and DSDSC algorithms have reached the optimum solution (920), also the other algorithms have better values in 500 iterations than 150 iterations. We conclude that the DSDSC and IPMFDS algorithms are better than GA to find the highest value with 150 iterations.

The version of GA are used here in this problem was bit string. We have made some changes in our algorithms: the size of fixed part of higher bits has decreased to a random value between 1and 5 in the dynamic schema, free dynamic schema and dynamic dissimilarity operators in DSDSC, DDS, FDS, MFDS, IPMFDS algorithms.

Table 7. 10 The results of knapsack problem with 50 items, for 20 runs of the DSC,DSDSC and GA algorithm (80 chromosomes) by using 150 and 500 iterations.

Algorithm name	Min value / Min time in seconds/ 150 It.	Max value/ Max time in seconds/ 150 It.	Mean values for all runs/ Average time/ 150 It.	Min value / Min time in seconds/ 500 It.	Max value/ Max time in seconds/ 500 It.	Mean values for all runs/ Average time/ 500 It.
GA	885	904	893	**904**	**920**	912
	0.312	0.521	0.38655	1.092	1.3112	1.1388
DSC	850	903	885	**888**	**920**	907
	0.81123	0.96721	0.88556	2.6208	3.0264	2.872
DSDSC	**878**	**915**	895	**898**	**920**	904
	0.88961	1.02081	0.9828	2.641	2.964	2.73
DDS	868	907	885	894	914	904
	0.81723	1.07231	0.88773	2.846	3.2108	2.9322
FDS	838	893	864	857	893	866
	1.025	1.1501	1.08566	3.0160	3.527	3.406
MFDS	848	902	878	876	913	899
	1.0967	1.2413	1.1825	2.979	3.441	3.198
IPMFDS	**866**	**912**	875	881	912	892
	1.0623	1.0915	1.0882	3.178	3.920	3.271

CHAPTER EIGHT: Conclusions

We see that our algorithms have ability to find the optimum solutions for two-dimensional functions in 100% success rate, but CMA-EA and DE haven’t this ability for some tested functions (like Easom, Rastrigin’s, Schwefel's).

In the IPMFDS algorithm, where the first initial population generates enough population diversity, and we use the different types of dynamic schema, free dynamic schema, similarity and dissimilarity operators with new random generation in each population, it is not a problem to find the global maximum/minimum solution, also random generation of a part of chromosomes supports population diversity in each iteration, especially with two-dimensional functions.

In free dynamic schema, when fixing the higher bits of each x_i, it means that we select the area to be searched for the best solution iteration after iteration, and when the fixed bits are increased, the search area will be more specific.

For ten-dimensional functions, we found it better to make a mask on some x_i and change the others completely, this idea gives a high percentage of success rate. Also we have seen that the CMA-ES doesn’t find the optimum solutions for the Zakharov, and Rastrigin’s functions in ten dimensions.

The fastest algorithm in terms of time was IPMFDS with two-dimensional functions but not with ten-dimensional functions. The fastest algorithm with ten-dimensional functions was DSDSC.

The lowest number of function evaluations was in the IPMFDS algorithm comparing with other algorithms for the most tested functions.

The dynamic schema, free dynamic schema and dynamic dissimilarity operators have big ability and possibility to reach the optimum solution.

In complex problems which had multiple local solutions, we discovered that, when the range of the function was reduced, the optimal solution was found faster.

In the knapsack problem we have observed that the DSDSC and DDS algorithms give the best solutions comparing to other algorithms (GA, DSC, FDS, MFDS, IPMFDS) in 150 iterations, in 500 iterations three algorithms (GA, DSC and DSDSC) reached to the optimum solution for the given data.

For the rotated and shifted functions that we have tested (taken from CEC 2017), we noted that the GA has not found the solutions for these functions; also CMA-ES has not found the best solution for one function (Rastrigin's), but our algorithms have found the optimum solution with different rates of success. We notice that for the first four algorithms (DSC, DSDSC, DDS, FDS) the success rates are low, but for the last two algorithms (MFDS, IPMFDS) we have reached 100% as a success rate for most tested functions.

We have applied the last two algorithms (MFDS, IPMFDS) to several 100-dimensional problems, and we notice the ability of these algorithms to solve these problems after making some changes in the algorithms, that is, copying higher bits from x_i to x_j in the best chromosome (where i and j are chosen randomly), while CMA-ES fails to find the solution for the Rastrigin function with d = 100. On the other hand, the CGA has ability to find the optimum solution by using bit string in the population type better than double vector with all tested functions with 100 dimensions.

Appendix A: Test Functions

A.1 Easom function [123], [124]

The Easom function is a unimodal test function, where the global minimum occurs in a small area relative to the search space. The function is used for minimization. It has two variables and the following definition:

$$f(x,y) = -cos(x)cos(y)exp(-(x-\pi)^2 + (y-\pi)^2))$$

The test area is usually restricted to the square $-100 \leq x \leq 100, -100 \leq y \leq 100$. Its global minimum is equal to $f(\pi,\pi) = -1$.

A.2 Matyas function [125]

This function has two variables and the following definition:

$$f(x,y) = 0.26(x^2 + y^2 - 0.48xy)$$

The test area is usually restricted to the square $-10 \leq x \leq 10, -10 \leq y \leq 10$.

The global minimum value at $f(0,0) = 0$.

A.3 Beale's function [126], [127]

This function has two variables and the following definition:

$$f(x,y) = (1.5 - x - xy)^2 + (2.25 - x + x\,y^2)^2 + (2.625 - x - xy^2)^2$$

The test area is usually restricted to the square $-4.5 \leq x \leq 4.5, -4.5 \leq y \leq 4.5$.

The global minimum value at $f(3,0.5) = 0$.

A.4 Booth's function [128]

This function has two variables and the following definition:

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$$

Test area is usually restricted to the square $-10 \le x \le 10,\ -10 \le y \le 10$.

The global minimum value at $f(1,3) = 0$.

A.5 Goldstein-Price function [123]

This function has two variables and the following definition:

$$f(x,y) = (1 + (x+y+1)^2\,(19 - 14x + 3x^2 - 14y + 6xy + 3y^2\,)) * (30 + (2x - 3y)^2\,(18 - 32x + 12x^2 + 48y - 36xy + 27y^2\,))$$

The test area is usually restricted to the square $-2 \le x \le 2\,, -2 \le y \le 2$. The global minimum value is equal $f(0,-1) = 3$ is obtainable for $(x,y) = (0,-1)$.

A.6 Schaffer function [129]

This function is defined in the search domain $x, y \in [-100,100]$, as follows:

$$f(x,y) = 0.5 + \frac{sin^2(x^2 - y^2) - 0.5}{(1 + 0.001(x^2 + y^2))^2}$$

and has the global min $f(0,0) = 0$.

A.7 Schwefel's function [123]

The Schwefel's function is misleading in that the best local minima are positioned far from the global minimum. Thus, the optimization algorithm may face an incorrect convergence. The function can be defined by the following equation:

$$f(x) = 418.9829 * n + \sum_{i=1}^{n} -x_i\ .sin\left(\sqrt{|x_i|}\right)$$

The test area is usually restricted to the hypercube $-500 \leq x_i \leq 500$, $i = 1, \ldots, n$. Its global minimum $f(420.9687, 420.9687, \ldots, 420.9687) = 0$.

In this work, this formula is used:

$$f(x) = 418.9829 * 2 + \sum_{i=1}^{2} -x_i \ . sin\left(\sqrt{|x_i|}\right)$$

for two dimensions and the minimum solution $f(x)$= 0 at (420.9687, 420.9687).

A.8 Branins's function [123]

The Branin function has two parameters and it can be considered as a global optimization assessment function. The function contains three global optima and can be defined according to the following equation:

$$f(x_1, x_2) = a \cdot (x_2 - b \cdot x^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot cos(x_1) + e$$

$$where \quad a = 1,\ b = \frac{5.1}{4 \cdot \pi^2}\ , c = \frac{5}{\pi}\ , d = 6\ , e = 10\ , f = \frac{1}{8 \cdot \pi}$$

It has three global minima equal to $f(x_1, x_2) = 0.397887$ and located as follows: $(x_1, x_2) = (-\pi, 12.275), (\pi, 2.275), (9.42478, 2.475)$.

A.9 Six-hump camel back function [123]

The Six-hump camel back function is basically a global optimization assessment function. This function possesses six local minima, inside the bounded area. Furthermore, two of the six minima are global. This function can be defined by the following equation:

$$f(x_1, x_2) = \left(4 - 2.1 x_1^{4/3}\right) \cdot x_1^2 + x_1 x_2 + (-4 + 4x_2^2) \cdot x_2^2$$

The test area is usually restricted to the rectangle $-3 \leq x_1 \leq 3$, $-2 \leq x_2 \leq 2$. Two global minima equal to $f(x) = -1.0316$ are located at $(x_1, x_2) = (-0.0898, 0.7126)$ and $(0.0898, -0.7126)$.

A.10 Shubert's function [123]

This is a multimodal test function. It has two variables and the following definition:

$$f(x_1, x_2) = \left(\sum_{i=1}^{5} i\ cos[(i+1)x_1 + i]\right) \cdot \left(\sum_{i=1}^{5} i\ cos[(i+1)x_2 + i]\right)$$

The test area is usually restricted to the square $-10 \leq x_1 \leq 10, -10 \leq x_2 \leq 10$.

It has eighteen global minimum equal to $f(x) = -186.7309$.

A.11 Martin and Gaddy function

This function has two variables and the following definition:

$$f(x_1, x_2) = (x_1 - x_2)^2 \cdot ((x_1 + x_2 - 10)/3)^2$$

The test area is usually restricted to the square $0 \leq x_1 \leq 10,\ 0 \leq x_2 \leq 10$, where the global minimum value at $f(5,5) = 0$.

A.12 Michalewicz function [123]

The Michalewicz functions is basically a multimodal testing function. It is defined as follows:

$$f(x_1, x_2) = 21.5 + x_1 \cdot sin(4\pi x_1) + x_2 \cdot sin(20\pi x_2)$$

The domain is $x_1 \in [-3,12.1]$, $x_2 \in [4.1,5.8]$ and the maximum value is at $f(11.631407, 5.724824) = 38.818208$ [130].

A.13 Holder table function [129]

The holder table function has multiple local minima with four global minima at:

$f(8.05502, 9.66458)$ *or* $f(8.05502, -9.66458)$ *or* $f(-8.05502, 9.66458)$ *or* $f(-8.05502, -9.66458)$ =-19.2085.

It is defined as follows:

$$f(x_1, x_2) = -\left|sin(x)\cos(x)\,exp\left(\left|1 + \frac{\sqrt{x_1^2 + x_2^2}}{\pi}\right|\right)\right|$$

A.14 Drop wave function [123]

This is a multimodal test function. This function has two variables and the following definition:

$$f(x_1, x_2) = -\frac{1 + \cos\left(12\sqrt{x_1^2 - x_2^2}\right)}{0.5(x_1^2 + x_2^2) + 2}$$

The test area is usually restricted to the square $-5.12 \leq x_1 \leq 5.12, -5.12 \leq x_1 \leq 5.12$.

A.15 Levy (#13) function [129], [125]

This function has two variables and the following definition:
$f(x_1, x_2) = sin^2(3\pi x_1) + (x_1 - 1)^2[1 + \sin(3\pi x_2)] + (x_2 - 1)^2[1 + \sin(2\pi x_2)]$,

where $x_1, x_2 \in [-10, 10]$, and the global minimum value is at $f(1, 1) = 0$.

A.16 Rastrigin’s function

The Rastrigin function has several local minima. It is highly multimodal, but locations of the minima are regularly distributed.

$$f(x) = 10 * d\sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i^2)]$$

The function is usually evaluated on the hypercube $x_i \in [-5.12, 5.12]$, for all $i = 1, \ldots, d$, where the global minimum value is at $f\ (0,0, \ldots ,0) = 0$.

A.17 Sum Squares function

The Sum Squares function also referred to as the Axis Parallel Hyper-Ellipsoid function, has no local minimum except the global one. It is continuous, convex and unimodal. It is defined as follows:

$$f(x) = \sum_{i=1}^{d} i x_i^2$$

The function is usually evaluated on the hypercube $x_i \in [-10, 10]$, for all $i = 1, \dots, d$, although this may be restricted to the hypercube $x_i \in [-5.12, 5.12]$, for all $i = 1, \dots, d$. The global minimum is at $f(0, \dots, 0) = 0$.

A.18 Sphere function

The Sphere function has d variables and the following definition. It is continuous, convex and unimodal.

$$f(x) = \sum_{i=1}^{d} x_i^2$$

The function is usually evaluated on the hypercube $x_i \in [-5.12, 5.12]$, for all $i = 1, \dots, d$. The global minimum is at $f(0, \dots, 0) = 0$.

A.19 Sum of different powers function

This function has d variables and the following definition. We have used d=10.

$$f(x_i) = \sum_{i=1}^{d} |x_i|^{i+1}$$

The function is usually evaluated on the hypercube $x_i \in [-1, 1]$, for all $i = 1, \dots, d$. The global minimum is at $f(0, \dots, 0) = 0$.

A.20 Ackley's function [123]

Ackley's function is a widely used multimodal test function. It has the following definition:

$$f(x) = -a \cdot \exp\left(-b \cdot \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d} \cos(c \cdot x_i)\right) + a + \exp(1)$$

It is recommended to set $a = 20, b = 0.2, c = 2\pi$. Test area is usually restricted to the hypercube $x_i \in [-32.768, 32.768]$, *i = 1, . . . ,d*. The global minimum is at $f(0, \dots, 0) = 0$.

A.21 Zakharov function

This function has d variables and the following definition. 10 dimensions have been used.

$$f(x) = \sum_{i=1}^{d} x_i^2 + (\sum_{i=1}^{d} 0.5 \cdot x_i)^2 + (\sum_{i=1}^{d} 0.5 \cdot x_i)^4$$

The function is usually evaluated on the hypercube $x_i \in [-5, 10]$, for all $i = 1, \dots, d$. The global minimum is at $f(0, \dots, 0) = 0$.

A.22 Rosenbrock's valley function [101]

This function has d variables and the following definition

$$f(x) = 10 * d\sum_{i=1}^{d-1} [\, 100\,(x_{i+1} - x_i^2) + (x_i - 1)^2]$$

The function is usually evaluated on the hypercube $x_i \in [-2.048, 2.048]$, for all $i = 1, \dots, d$. The global minimum is at $f(1, \dots, 1) = 0$.

Shifted and rotated test functions (CEC 2017 benchmark) [45]

The test functions in this part are shifted by o and rotated by M where

o: shifted global optimum which is randomly distributed in [-80,80],

M: rotation matrix.

A.23 Shifted and Rotated Bent Cigar

$$F_1(x) = f_1(M(x-o)) + F_1^*$$

where f_1 is

$$f_1(x) = x_1^2 + 10^6 \sum_{i=2}^{d} x_i^2$$

The properties of this function are: unimodal, non-separable, smooth but narrow ridge.

A.24 Shifted and Rotated Sum of Different Power Function

$$F_2(x) = f_2(M(x-o)) + F_2^*$$

where f_2 is

$$f_2(x_i) = \sum_{i=1}^{d} |x_i|^{i+1}$$

The properties of this function are: unimodal, non-separable, symmetric.

A.25 Shifted and Rotated Zakharov Function

$$F_3(x) = f_3(M(x-o) + F_3^*$$

where f_3 is

$$f_3(x) = \sum_{i=1}^{d} x_i^2 + (\sum_{i=1}^{d} 0.5 \cdot x_i)^2 + (\sum_{i=1}^{d} 0.5 \cdot x_i)^4$$

The properties of this function are: unimodal, non-separable.

A.26 Shifted and Rotated Rosenbrock's Function

$$F_4(x) = f_4\left(M\left(\frac{2.048(x-o)}{100}\right)+1\right)+F_4^*$$

where f_4 is

$$f_4(x) = 10 * d\sum_{i=1}^{d-1}[\,100\,(x_{i+1} - x_i^2) + (x_i - 1)^2]$$

The properties of this function are: multi-modal, non-separable, the number of local optima is huge.

A.27 Shifted and Rotated Rastrigin's Function

$$F_5(x) = f_5(M(x-o)) + F_5^*$$

where f_4 is

$$f_5(x) = 10 * d\sum_{i=1}^{d}[x_i^2 - 10\cos(2\pi x_i^2)]$$

The properties of this function are: multi-modal, non-separable, the number of local optima is huge and second best local optimum is far from the global optimum.

References

[1] Y. Xinjie and G. Mitsuo, *Introduction to Evolutionary Algorithms*. Springer-Verlag London Limited, 2010.

[2] E. K.P. Chong and S. H. ZAK, *An Introduction to Optimization*. Wiley, USA, 2001.

[3] R. Subramani and C. Vijayalakshmi, "A review on advanced optimization techniques," *ARPN J. Eng. Appl. Sci.*, vol. 11, no. 19, pp. 11675–11683, 2016.

[4] T. El-Ghazali, *Metaheuristics From Design to Implementation*. JohnWiley & Sons, Inc., Hoboken, New Jersey, Canada, 2009.

[5] F. Glover, "Tabu Search - Part I," *ORSA J. Comput.*, vol. 1, no. 3, pp. 190–206, 1989.

[6] F. Glover, "Tabu search - Part II," *ORSA J. Comput.*, vol. 2, no. 1, pp. 4–32, 1990.

[7] S. Ólafsson, "Metaheuristics," *Handbooks Oper. Res. Manag. Sci.*, pp. 633–654, 2006.

[8] K. Amouzgar, "Multi-Objective Optimization using Genetic Algorithms," *Thesis Work. Tek. Hogsk.*, p. 79, 2012.

[9] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, New York, 2001.

[10] S. S. Rao, *Engineering Optimization: Theory and Practice*. John Wiley & Sons, Inc., 2009.

[11] H. Ishibuchi and Y. Nojima, "Optimization of scalarizing functions through evolutionary multiobjective optimization," *Lect. Notes Comput. Sci. 4403 Evol. Multi-Criterion Optim. - EMO , Springer, Berlin*, pp. 51–65, 2007.

[12] G. Venturelli, E. Benini, and Ł. Łaniewski-Wołłk, "A kriging-assisted multiobjective evolutionary algorithm," *Appl. Soft Comput.*, vol. 58, pp. 155–175, 2017.

[13] S. Bandyopadhyay and S. Saha, *Unsupervised Classification, chapter 2, Some*

Single- and Multiobjective Optimization techniques. Springer-Verlag Berlin Heidelberg., 2013.

[14] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1999.

[15] J. D. Farmer, N. H. Packard, and A. S. Perelson, "The immune system, adaptation, and machine learning," *Physica*, vol. 22, no. 1–3, pp. 187–204, 1986.

[16] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, Springer N., vol. 146. Springer New York Dordrecht Heidelberg London, 2010.

[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," *1995 IEEE Int. Conf. Neural Networks (ICNN 95)*, vol. 4, pp. 1942–1948, 1995.

[18] R. Storn and K. Price, "Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces," *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[19] D. Mora-Melia, P. L. Iglesias-Rey, F. J. Martinez-Solano, and P. Munoz-Velasco, "The efficiency of setting parameters in a modified shuffled frog leaping algorithm applied to optimizing water distribution networks," *Water (Switzerland)*, vol. 8, no. 182, p. 14, 2016.

[20] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," *Tech. report-TR06, Erciyes Univ. Eng. Fac. Comput. Eng. Dep. Kayseri/Türkiye*, vol. technical, 2005.

[21] R. V. Rao and V. J. Savsani, *Mechanical Design Optimization Using Advanced Optimization Techniques, Chapter 2*. Springer-Verlag London, 2012.

[22] J. Liu and L. Tang, "A modified genetic algorithm for single machine scheduling," *Comput. Civ. Infrastruct. Eng.*, vol. 37, pp. 43–46, 1999.

[23] C. Preechakul and S. Kheawhom, "Modified genetic algorithm with sampling techniques for chemical engineering optimization," *J. Ind. Eng. Chem.*, vol. 15, no. 1, pp. 110–118, 2009.

[24] H. Cui and O. Turan, "Application of a new multi-agent hybrid co-evolution based particle swarm optimisation methodology in ship design," *CAD Comput. Aided*

Des., vol. 42, no. 11, pp. 1013–1027, 2010.

[25] I. Montalvo, J. Izquierdo, R. Pérez-García, and M. Herrera, "Improved performance of PSO with self-adaptive parameters for computing the optimal design of Water Supply Systems," *Eng. Appl. Artif. Intell.*, vol. 23, no. 5, pp. 727–735, 2010.

[26] A. R. Yildiz, "A novel particle swarm optimization approach for product design and manufacturing," *Int. J. Adv. Manuf. Technol.*, vol. 40, no. 5–6, pp. 617–628, 2009.

[27] H. Yu, G. Gu, H. Liu, J. Shen, and J. Zhao, "A modified ant colony optimization algorithm for tumor marker gene selection," *Genomics. Proteomics Bioinformatics*, vol. 7, no. 4, pp. 200–208, 2009.

[28] Q. Shen, J.-H. Jiang, J.-C. Tao, G.-L. Shen, and R.-Q. Yu, "Modified ant colony optimization algorithm for variable selection in QSAR modeling: QSAR studies of cyclooxygenase inhibitors.," *J. Chem. Inf. Model.*, vol. 45, no. 4, pp. 1024–9, 2005.

[29] D. Karaboga and B. Akay, "A modified Artificial Bee Colony (ABC) algorithm for constrained optimization problems," *Appl. Soft Comput. J.*, vol. 11, no. 3, pp. 3021–3031, 2011.

[30] S. Sajeevan and N. Padmavathy, "Optimal allocation and sizing of distributed generation using artificial bee colony Algorithm," *Int. Res. J. Eng. Technol.*, vol. 3, no. 2, pp. 1–11, 2016.

[31] W. Mao, H.-Y. Lan, and H.-R. Li, "A new modified artificial bee colony algorithm with exponential function adaptive steps," *Comput. Intell. Neurosci.*, vol. 2016, no. i, p. 13, 2016.

[32] B. Tang, Z. Zhu, and J. Luo, "Hybridizing particle swarm optimization and differential evolution for the mobile robot global path planning," *Int. J. Adv. Robot. Syst.*, vol. 13, no. 3, p. 17, 2016.

[33] W. Y. Lin, "A GA-DE hybrid evolutionary algorithm for path synthesis of four-bar

linkage," *Mech. Mach. Theory*, vol. 45, no. 8, pp. 1096–1107, 2010.

[34] Y. P. Chang, "An ant direction hybrid differential evolution algorithm in determining the tilt angle for photovoltaic modules," *Expert Syst. with Appl. Elsevier*, vol. 37, no. 7, pp. 5415–5422, 2010.

[35] Y. Marinakis and M. Marinaki, "A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem," *Comput. Oper. Res.*, vol. 37, no. 3, pp. 432–442, 2010.

[36] V. Savsani, "HBBABC: A hybrid optimization algorithm combining Biogeography Based Optimization (BBO) and Artificial Bee Colony (ABC) optimization for obtaining global solution of discrete design problems," *I nternational J. Comput. Eng. Res.*, vol. 2, no. 7, pp. 85–97, 2012.

[37] Y.-T. Kao and E. Zahara, "A hybrid genetic algorithm and particle swarm optimization for multimodal functions," *Appl. Soft Comput.*, vol. 8, no. 2, pp. 849–857, 2008.

[38] L. Shi, C. Chen, and E. Yucesan, "Simultaneous simulation experiments and nested partition for discrete resource allocation in supply chain management," *Proc. Winter Simul. Conf.*, pp. 395–401, 1999.

[39] L. Shi and S. Olafsson, "Two-stage nested partitions method for stochastic optimization," *Methodol. Comput. Appl. Probab.*, vol. 2, no. 3, pp. 271–291, 2000.

[40] I. Younas, "Using Genetic Algorithms for Large Scale Optimization of Assignment , Planning and Rescheduling Problems," Doctoral Thesis in Electronics and Computer Systems Stockholm, Sweden, 2014.

[41] J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Second edition: MIT Press, 1975.

[42] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*. Springer-Verlag Berlin Heidelberg, 2008.

[43] K. F. Man, K. S. Tang, and S. Kwong, "Genetic algorithms : Concepts and

applications," *IEEE Trans. Ind. Electron.*, vol. 43, no. 5, pp. 519–534, 1996.

[44] S. A.-H. Soliman and A.-A. H. Mantawy, *Modern Optimization Techniques with Applications in Electric Power Systems, chapter 2, Mathematical Optimization Techniques*. Springer, 2012.

[45] N. H. Awad, M. Z. Ali, P. N. Suganthan, J. J. Liang, and B. Y. Qu, "Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization," *http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2017/CEC2017.htm*, 2017.

[46] K. Manda, S. C. Satapathy, and B. Poornasatyanarayana, "Population based meta-heuristic techniques for solving optimization problems : A selective survey," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 11, pp. 206–211, 2012.

[47] F. Glover, "PATHS FOR INTEGER PROGRAMMING," *Comput. Ops. Res.*, vol. 13, no. 5, pp. 533–549, 1986.

[48] I. H. Osman and G. Laporte, "Metaheuristics: A bibliography," *Ann. Oper. Res.*, vol. 63, no. 5, pp. 511–623, 1996.

[49] G. P. Rajappa, "Solving Combinatorial Optimization Problems Using Genetic Algorithms and Ant Colony Optimization," 2012.

[50] A. Fita, "Metaheuristic start for gradient based optimization algorithms," *Am. J. Comput. Appl. Math.*, vol. 5, no. 3, pp. 88–99, 2015.

[51] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci. Elsevier*, vol. 237, pp. 82–117, 2013.

[52] A. Cruz-bernal, "Meta-Heuristic optimization techniques and its applications in robotics," *INTECH, World Larg. Sci. , Technol. Med. Open Access B. Publ.*, pp. 53–75, 2013.

[53] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Heidelberg New York Dordrecht London, 2015.

[54] P. Chang, Y. Wang, and C. Liu, "New operators for faster convergence and better

solution quality in modified genetic algorithm," *Wang L., Chen K., Ong Y.S. Adv. Nat. Comput. ICNC 2005. Lect. Notes Comput. Sci. vol 3611. Springer, Berlin, Heidelb.*, pp. 983–991, 2005.

[55] G. Q. Zeng, K. Di Lu, J. Chen, Z. J. Zhang, Y. X. Dai, W. W. Peng, and C. W. Zheng, "An improved real-coded population-based extremal optimization method for continuous unconstrained optimization problems," *Math. Probl. Eng.*, vol. 2014, pp. 1–10, 2014.

[56] I. C. D. Informatik, C. Dortmund, and G. Rudolph, "Convergence of Evolutionary Algorithms in General Search Spaces," pp. 0–4, 1996.

[57] G. U. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 96–101, 1994.

[58] J. Dréo, P. Alain, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization*. Springer-Verlag Berlin Heidelberg, 2006.

[59] M. D. Vose, "Logarithmic convergence of random heuristic search." Evolutionary Computation, pp. 395–404, 1997.

[60] C. C. Y. Dorea, J. A. G. Jr, R. Morgado, and G. C. Andre, "Multistage Markov chain modeling of the genetic algorithm and convergence results," *Numer. Funct. Anal. Optim.*, vol. 2, no. 31, pp. 163–171, 2010.

[61] A. G. C. Pereira and V. S. M. Campos, "Multistage non homogeneous Markov chain modeling of the non homogeneous genetic algorithm and convergence results," *Commun. Stat. - Theory Methods*, vol. 45, no. 6, pp. 1794–1804, 2016.

[62] J. A. Rojas Cruz and A. G. C. Pereira, "The elitist non-homogeneous genetic algorithm: Almost sure convergence," *Stat. Probab. Lett.*, vol. 83, no. 10, pp. 2179–2185, 2013.

[63] J. A. R. Cruz and I. C. Diniz, "Mean convergence time of inhomogeneous genetic algorithm with elitism," *Numer. Funct. Anal. Optim.*, vol. 37, no. 8, pp. 966–974, 2016.

[64] R. R. Sharapov and A. V. Lapshin, "Convergence of genetic algorithms," *Pattern*

Recognit. Image Anal., vol. 16, no. 3, pp. 392–397, 2006.

[65] M. Studniarski, "Stopping criteria for a general model of genetic algorithm," *Pr. Nauk. Politech. Warszawaskiej*, vol. 169, no. 4, pp. 1–8, 2009.

[66] M. Studniarski, "Stopping criteria for genetic algorithms with application to multiobjective optimization," *PPSN XI, Part I, LNCS 6238, Springer-Verlag Berlin Heidelb.*, pp. 697–706, 2010.

[67] M. Studniarski, "Finding all minimal elements of a finite partially ordered set by genetic algorithm with a prescribed probabilty," *Numer. Algebr. Control Optim.*, vol. 1, no. 3, pp. 389–398, 2011.

[68] S. Y. Yuen, C. K. Fong, and H. S. Lam, "Guaranteeing the probability of success using repeated runs of genetic algorithm," *Image Vis. Comput.*, vol. 19, pp. 551–560, 2001.

[69] S. Yin, H. Shan, C. Ki, S. Feng, and C. Kin, "A robust iterative hypothesis testing design of the repeated genetic algorithm," *Image Vis. Comput.*, vol. 23, pp. 972–980, 2005.

[70] E. Kita and T. Maruyama, *Genetic Algorithm Based on Schemata Theory, chapter 3 from the book Evolutionary Algorithms*. InTech, 2011.

[71] S. Tsutsui and Y. Fujimoto, "Forking genetic algorithm with blocking and shrinking modes (fGA)," *Proc. 5th Int. Conf. Genet. Algorithms*, pp. 206–215, 1993.

[72] D. E. Goldberg, "Messy Genetic Algorithms : Motivation , Analysis , and First Results," *Complex Syst.*, vol. 3, pp. 493–530, 1989.

[73] S. Forrest and M. Mitchell, "What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation," *Mach. Learn.*, vol. 13, no. 2, pp. 285–319, 1993.

[74] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "A novel population initialization method for accelerating evolutionary algorithms," *Comput. Math. with Appl.*, vol. 53, no. 10, pp. 1605–1614, 2007.

[75] H. Maaranen, K. Miettinen, and A. Penttinen, “On initial populations of a genetic algorithm for continuous optimization problems,” *J. Glob. Optim.*, vol. 37, no. 3, pp. 405–436, 2007.

[76] B. Kazimipour, X. Li, and A. K. Qin, “A Review of Population Initialization Techniques for Evolutionary Algorithms,” *IEEE Congr. Evol. Comput.*, pp. 2585–2992, 2014.

[77] J. Arabas, Z. Michalewicz, and J. Mulawka, “GAVaPS - genetic algorithm with varying population size,” in *First IEEE Conf. on Evolutionary Computation, Piscataway, NJ, 1994. IEEE Press.*, 1994, pp. 73–78.

[78] O. Roeva, “A modified genetic algorithm for parmeter identification of frementation processes,” *Biotechnol. Biotechnol. Equip.*, vol. 20, no. 1, pp. 202–209, 2006.

[79] M. Bessaou and P. Siarry, “A genetic algorithm with real-value coding to optimize multimodal continuous functions,” *Struct. Multidiscip. Optim.*, vol. 23, pp. 63–74, 2001.

[80] N. Patel and N. Padhiyar, “Modified genetic algorithm using Box Complex method : Application to optimal control problems,” *J. Process Control*, vol. 26, pp. 35–50, 2015.

[81] J. Arabas, *Wyklady z Algorytmow Ewolucyjnych (Lectures on Evolutionary Algorithms)*. Warszawa: Wydawnictwa Naukowo-Techniczne, 2001.

[82] X. Han, Y. Liang, Z. Li, G. Li, X. Wu, B. Wang, G. Zhao, and C. Wu, “An efficient genetic algorithm for optimization problems with time-consuming fitness evaluation,” *Int. J. Comput. Methods*, vol. 12, no. 1, p. 1350106 (24 pages), 2015.

[83] Y. Yu and Z. Zhou, “A new approach to estimating the expected first hitting time of evolutionary algorithms,” *Artif. Intell.*, vol. 172, no. 15, pp. 1–48, 2008.

[84] N. J. Radcliffe, “The algebra of genetic algorithms,” *Ann. Math. Artif. Intell.*, vol. 10, no. 4, pp. 339–384, 1994.

[85] Z. Yuan and L. I. Bingfen, “The empirical study of the schema theory of genetic

algorithm based on 3-satisfiability problem," *Jt. Int. Mech. Electron. Inf. Technol. Conf.*, pp. 448–453, 2015.

[86] M. M. Lankhorst, "Genetic algorithms in data analysis," *Groningen : s.n.*, 1996.

[87] Michalewicz Zbigniew, *Genetic Algorithms + Data Structures = Evolution Programs*, (3ed). Springer series Artificial Intelligence, 1996.

[88] A. B. M. Sultan, R. Mahmod, M. N. Sukaiman, and M. R. Abu Bakar, "Maintaining diversity for genetic algorithm: a case of timetabling problem," *J. Teknol. Malaysia*, vol. 44, no. D, pp. 123–130, 2006.

[89] M. Lewchuk, "Genetic invariance: A new type of genetic algorithm," *Tech. Rep. TR 92-05, Dept. Comput. Sci. Univ. Alberta*, 1992.

[90] M. G. C. Resende and J. P. de Sousa, *Metaheuristics: Computer-Decision Making*. Springer Science+ Business Media, LL, 2004.

[91] M. Studniarski, R. Al-jawadi, and A. Younus, "An evolutionary optimization method based on scalarization for multi-objective problems," *Borzemski L., Świątek J., Wilimowska Z. Inf. Syst. Archit. Technol. Proc. 38th Int. Conf. Inf. Syst. Archit. Technol. – ISAT 2017. Adv. Intell. Syst. Comput. Spri*, vol. 656, pp. 48–58, 2018.

[92] E. Rahmo and M. Studniarski, "A new global scalarization method for multiobjective optimization with an arbitrary ordering cone," *Appl. Math.*, vol. 8, pp. 154–163, 2017.

[93] "CMA-ES. Matlab program." URL https://www.mathworks.com/ matlabcentral/fileexchange/52898-cma-es-in-matlab.

[94] K. V. Price, "Differential Evolution," *http://www.dii.unipd.it/~alotto/didattica/corsi/Elettrotecnica%20computazionale/ DE.pdf*, pp. 187–214, 2013.

[95] "How the Genetic Algorithm Works - MATLAB & Simulink," *http://www.mathworks.com/help/gads/how-the-genetic-algorithm-works.html*. .

[96] J. G. Kemeny, J. L. Snell, and A. W. Knapp, *Denumerable Markov Chains*.

Graduate Texts in Mathematics. Springer New York Dordrecht Heidelberg London, 1976.

[97] Y. Wu, G. Sun, K. Su, L. Liu, H. Zhang, B. Chen, and M. Li, “Dynamic self-adaptive double population particle swarm optimization algorithm based on lorenz equation,” *J. Comput. Commun.*, vol. 5, no. 13, pp. 9–20, 2017.

[98] T. Park and K. R. Ryu, “A dual-population genetic algorithm for adaptive diversity control,” *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 865–884, 2010.

[99] R. Al-Jawadi, “An optimization algorithm based on dynamic schema with dissimilarities and similarities of chromosomes,” *Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 7, no. 8, pp. 1278–1285, 2016.

[100] R. Al-jawadi, M. Studniarski, and A. Younus, “New genetic algorithm based on dissimilaries and similarities,” *Comput. Sci. Journal, AGH Univ. Sci. Technol. Pol.*, vol. 19, no. 1, p. 19, 2018.

[101] A. S. Eesa, A. M. A. Brifcani, and Z. Orman, “A new tool for global optimization problems- cuttlefish algorithm,” *Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 8, no. 9, pp. 1198–1202, 2014.

[102] R. Al-Jawadi and M. Studniarski, “An Optimization Algorithm Based on Multi-Dynamic Schema of Chromosomes,” *Int. Conf. Artif. Intell. Soft Comput. Springer, Cham.*, vol. 10841, pp. 279–289, 2018.

[103] M. Amiri and M. Khajeh, “Developing a bi-objective optimization model for solving the availability allocation problem in repairable series – parallel systems by NSGA II,” *J. Ind. Eng. Int.*, vol. 12, no. 1, pp. 61–69, 2016.

[104] J. Safari, “Multi-objective reliability optimization of series-parallel systems with a choice of redundancy strategies,” *Reliab. Eng. Syst. Saf.*, vol. 108, no. 1, pp. 10–20, 2012.

[105] M. Aghaei, A. Zeinal, H. Mostafa, and A. Ardakan, “Redundancy allocation problem for k -out-of- n systems with a choice of redundancy strategies,” *J. Ind. Eng. Int.*, vol. 13, no. 1, pp. 81–92, 2017.

[106] S. Gotshall and B. Rylander, “Optimal population size and the genetic algorithm,” *Proc. Genet. Evol. Comput. Conf.*, pp. 1–5, 2000.

[107] O. Roeva, “Influence of the population size on the genetic algorithm performance in case of cultivation process modelling,” *Proc. 2013 Fed. Conf. Comput. Sci. Inf. Syst. pp.*, pp. 371–376, 2013.

[108] V. R. P and M. B. V, “Improving the performance of genetic algorithm by reducing the population size,” *I nternational J. Comput. Eng. Res.*, vol. 3, no. 8, pp. 86–91, 2013.

[109] T. Friedrich and P. S. Oliveto, “Analysis of Diversity-Preserving Mechanisms for Global Exploration,” *Evol. Comput.*, vol. 17, no. 4, pp. 455–476.

[110] D. Gupta and S. Ghafir, “An overview of methods maintaining diversity in genetic algorithms,” *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 5, pp. 56–60, 2012.

[111] C. Elegbede and K. Adjallah, “Availability allocation to repairable systems with genetic algorithms : a multi-objective formulation,” *Reliab. Eng. Syst. Saf.*, vol. 82, no. 1, pp. 319–330, 2003.

[112] H. Zoulfaghari, A. Z. Hamadani, and M. A. Ardakan, “Bi-objective redundancy allocation problem for a system with mixed repairable and non-repairable components,” *ISA Trans.*, vol. 53, no. 1, pp. 17–24, 2014.

[113] K. Deb, A. Member, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[114] F. A. Badroo, “The applicability of genetic algorithm in cryptanalysis : A survey,” *Int. J. Comput. Appl. (0975 – 8887)*, vol. 130, no. 9, pp. 42–46, 2017.

[115] T. Soule, A. Piszcz, and T. Soule, “Genetic programming : Optimal population sizes for varying complexity problems,” *GECCO’06, Seattle, Washington, USA, Conf.*, 2006.

[116] D. Hougen, “Initial population for genetic algorithms : A metric approach,” *Proc. 2007 Int. Conf. Genet. Evol. Methods, GEM 2007, June 25-28, 2007, Las Vegas,*

Nevada, USA, Hamid R. Arab. Jack Y. Yang Mary Qu Yang (Eds), pp. 43-49, CSREA Press., 2007.

[117] P. A. Diaz-gomez and D. F. Hougen, “Empirical study : Initial population diversity and genetic algorithm performance,” *Artif. Intell. Pattern Recognit.*, pp. 334–341, 2007.

[118] E. Stripling, S. vanden Broucke, and B. Baesens, “Solving the Knapsack Problem with a Simple Genetic Algorithm.” Data Science Briefings, the DataMiningApps newsletter. http://www.dataminingapps.com/2017/03/solving-the-knapsack-problem-with-a-simple-genetic-algorithm/, 2017.

[119] U. Arıkan, “L2 : Algorithms : Knapsack Problem & BnB,” people.sutd.edu.sg/~ugur_arikan/Documents/Tutorials/.../L2.pdf, 2016.

[120] Z. Michalewicz and J. Arabas, “Genetic Algorithms for the 0 / 1 Knapsack Problem,” in *Raś Z.W., Zemankova M. (eds) Methodologies for Intelligent Systems. ISMIS 1994. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), Springer*, 1994, vol. 869, pp. 134–143.

[121] C. Queen, “Genetic Algorithms Applied to the Knapsack Problem,” math.stmarys-ca.edu/wp-content/uploads/.../Christopher-Queen.pdf, 2016.

[122] S. Sadeghyan, “solving 0-1knapsack problem by using Genetic Algorithm matlab.” https://github.com/5amron/solving-0-1-knapsack-problem-by-using-Genetic-Algorithm-matlab, https://github.com/5amron/solving-0-1-knapsack-problem-by-using-Genetic-Algorithm-matlab, 2017.

[123] M. Molga and C. Smutnicki, “Test functions for optimization needs,” *Comput. Inform. Sci.*, pp. 1–43, 2005.

[124] H. Pohlheim, “Examples of objective functions,” *GEATbx Examples*, 2007.

[125] K. Chwastek, M. Najgebauer, and J. Szczyglowski, “Performance of some novel optimization algorithms,” *Prz. Elektrotechniczny*, vol. 88, no. 12, pp. 191–193, 2012.

[126] M. Vali, “Rotational mutation genetic algorithm on optimization problems,” *arXiv*

Prepr. arXiv1307.5838. Jul 22, 2013.

[127] V. Seksaria, "Multimodal optimization by sparkling squid populations," *arXiv Prepr. arXiv1401.0858. 2014 Jan 5*, 2014.

[128] S. E. K. Fateen and A. Bonilla-Petriciolet, "Gradient-based cuckoo search for global optimization," *Math. Probl. Eng.*, vol. 2014, p. 12, 2014.

[129] S. K. Mishra, "Some new test functions for global optimization and performance of repulsive particle swarm method," *https//ssrn.com/abstract=926132 or http//dx.doi.org/10.2139/ssrn.926132*, 2006.

[130] M. Gen, R. Cheng, and L. Lin, *Network Models and Optimization : Multiobjective GA Approach*. Springer Science & Business Media, 2009.