Numerical optimization using differential evolution

Awad, Noor Hussien Ali

2019

Awad, N. H. A. (2019). Numerical optimization using differential evolution. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/83265>

<https://doi.org/10.32657/10220/48011>

**NUMERICAL OPTIMIZATION USING DIFFERENTIAL
EVOLUTION**

NOOR HUSSIEN ALI AWAD

**SCHOOL OF ELECTRICAL & ELECTRONIC
ENGINEERING
2019**

NUMERICAL OPTIMIZATION USING DIFFERENTIAL EVOLUTION

NOOR HUSSIEN ALI AWAD

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarized materials, and has not been submitted for a higher degree to any other University or Institution.

18/3/2019
Date

[Noor Hussien Ali Awad]

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

18 Mar 2019

Date

[Ponnuthurai Nagaratnam Suganthan]

Authorship Attribution Statement

This thesis contains material from [6] paper(s) published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Edward Jaser, Differential evolution with stochastic fractal search algorithm for global numerical optimization, in Proc. IEEE Congr. Evol. Comput., CEC 2016, pp. 3154-3161.

The contributions of the co-authors are as follows:

- A/Prof Suganthan provided the initial project directions and edited the manuscript drafts.
- I prepared the manuscript drafts. The manuscript was revised by Dr. Ali and Dr. Jaser
- I implemented the algorithm and performed all the experimental study. I also analyzed the results.

Chapter 4 is published as Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Edward Jaser, A decremental stochastic fractal differential evolution for global numerical optimization, Inf. Sci., vol. 372, pp. 470-491, 2016.

The contributions of the co-authors are as follows:

- A/Prof Suganthan suggested the initial project directions and edited the manuscript drafts.
- I wrote the drafts of the manuscript. The manuscript was revised by Dr. Ali and Dr. Jaser
- I designed the algorithm and did the coding stage. I also performed all the experimental work and analyzed the results.

Chapter 6 is published as Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N.

Suganthan, Robert G. Reynolds, An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CEC2014 benchmark problems, in Proc. IEEE Congr. Evol. Comput., CEC 2016, pp. 2958-2965.

The contributions of the co-authors are as follows:

- A/Prof Suganthan edited the manuscript drafts.
- I suggested the initial project directions together with Dr. Ali and designed the model
- I wrote the drafts of the manuscript. The manuscript was revised by Dr. Ali and Dr. Reynolds.
- I coded the algorithm and did the implementation stage. I also performed all the experimental work and analyzed the results.

Chapter 7 is published as Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Ensemble of parameters in a sinusoidal differential evolution with niching-based population reduction, Swarm and Evolutionary Computation, vol. 39, pp. 141-156, 2017.

The contributions of the co-authors are as follows:

- A/Prof Suganthan suggested the initial directions of the project and edited the manuscript.
- I designed the algorithm, coded it and wrote the drafts of the manuscript. I also performed all the experimental work and analyzed the results
- The manuscript was revised by Dr. Ali.

Chapter 9 is published as Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Robert G. Reynolds, Ali M. Shatnawi, A novel differential crossover strategy based on covariance matrix learning with Euclidean neighborhood for solving real-world problems, in Proc. IEEE Congr. Evol. Comput., CEC 2017, pp. 380-386.

The contributions of the co-authors are as follows:

project and edited the manuscript.

- I wrote the drafts of the manuscript. I also performed all the experimental work and analyzed the results.
- Dr. Reynolds suggested experimental benchmark.
- The manuscript was revised by Dr. Ali and Dr. Reynolds.

Chapter 10 is published as Noor H. Awad, Mostafa Z. Ali, Rammohan Mallipeddi, Ponnuthurai N. Suganthan, An Improved Differential Evolution Algorithm using Efficient Adapted Surrogate Model for Numerical Optimization. Inf. Sci, vols. 451–452, pp. 326-347, 2018.

The contributions of the co-authors are as follows:

- A/Prof Suganthan and Dr. Rammohan suggested the initial directions of the project and edited the manuscript.
- I designed the algorithm and did the coding stage. I also performed all the experimental work and analyzed the results.
- I wrote the drafts of the manuscript.
- The manuscript was revised by Dr. Ali and Dr. Rammohan.

Date

18/3/2019

[Noor Hussien Ali Awad]

Acknowledgments

I would like to express my sincere gratitude to my supervisor Prof. Ponnuthurai Nagaratnam Suganthan for motivation, guidance and support throughout my research program and for the opportunity to undertake this study work at Nanyang Technological University. I feel honoured to be able to work under his guidance. I also would like to express my appreciation to Dr. Mostafa Ali, my past supervisor in my master's degree, for his guidance and support throughout my research work. His assistance has been invaluable to the achievements in my research journey in the past four years.

I also would like to thank the Agency of Science, Technology and Research (A*STAR) and Nanyang Technological University for Singapore International Graduate Award (SINGA), funding and scholarship.

Finally, I would like to thank my family: my father, my brothers and sisters for supporting me spiritually throughout this study and supporting me in my whole life in general. And last but not least, all I am I owe to my mother, who rest in peace. I attribute all my success in life to the moral, intellectual and physical education I received from her. This one is for you mom!

Table of Contents

Statement of Originality	i
Supervisor Declaration Statement.....	ii
Authorship Attribution Statement	iii
Acknowledgments	vi
Table of Contents	vii
List of Figures.....	xii
List of Pseudocodes	xiii
List of Tables	xiv
Abbreviations	xvii
Abstract.....	1
Chapter 1 Introduction.....	2
1.1 Motivations and Objectives	2
1.2 Major Contribution of the Thesis	4
1.3 Organization of the Thesis.....	6
Chapter 2 Research Background and Literature Review	9
2.1 Numerical Optimization	9
2.2 Canonical Differential Evolution.....	11
2.3 Literature Review	16
2.3.1 A Review of some state-of-the-art DE Algorithms	18
2.3.2 Other Improved DE Algorithms	26
2.4 Summary.....	29
PART I Improved Hybrid Differential Evolution Algorithms	32
Chapter 3 Hybrid differential evolution and stochastic fractal search:	
The SFS-DP_{DE-GW} algorithm	33
3.1 Introduction	33
3.2 Stochastic Fractal Search.....	35
3.3 SFS-DP _{DE-GW} : How it Works	39

3.3.1	Differential Evolution Diffusion Process.....	39
3.3.2	Putting it all together.....	40
3.4	Experimental Results.....	41
3.4.1	Numerical Benchmark	41
3.4.2	Parametric Setup and Algorithm Complexity.....	42
3.4.3	Compared Algorithms.....	43
3.4.4	Results on Numerical Benchmark	44
3.5	Conclusion.....	46
Chapter 4 A Decremental Stochastic Fractal Differential Evolution:		
The <i>d</i>DSF-EA algorithm.....		47
4.1	Introduction	47
4.2	<i>d</i> DSF-EA: How it Works	48
4.2.1	Success-based Update Process.....	50
4.2.2	Population Size Dynamic Reduction	51
4.2.3	Putting it all together.....	52
4.3	Experimental Results.....	52
4.3.1	Parametric Setup and Algorithm Complexity.....	52
4.3.2	Compared Algorithms.....	53
4.3.3	Results on Numerical Benchmark	54
4.4	Conclusion.....	57
Chapter 5 Summary of Part I.....		58
5.1	Comprehensive Comparison of the Methods in Part I	59
PART II Improved Differential Evolution Algorithms based on		
Ensemble Parameter Adaptation		65
Chapter 6 Ensemble sinusoidal differential evolution: The LSHADE-		
EpSin algorithm		66
6.1	Introduction	66
6.2	LSHADE-EpSin: How it Works.....	68

6.2.1	Ensemble of Parameter Adaptation	68
6.2.2	Local Search.....	71
6.2.3	Putting it all together.....	72
6.3	Experimental Results	73
6.3.1	Numerical Benchmark	73
6.3.2	Parametric Setup and Algorithm Complexity	73
6.3.3	Compared Algorithms.....	74
6.3.4	Experiments and numerical results	75
6.4	Conclusion	76
Chapter 7 Ensemble sinusoidal differential evolution algorithm with niching reduction: EsDEr-NR algorithm.....		78
7.1	Introduction	78
7.2	EsDEr-NR: How it Works	79
7.2.1	Niching-based Population Size Reduction.....	80
7.2.2	Putting it all together.....	83
7.3	Experimental Results	83
7.3.1	Parametric Setup and Algorithm Complexity	84
7.3.2	Compared Algorithms.....	84
7.3.3	Results on Numerical Benchmark	86
7.4	Conclusion	90
Chapter 8 Summary of Part II		91
8.1	Overall Comparison of the Methods in Part I-II.....	92
PART III Applications: Improved Differential Evolution Algorithms for Solving Real-World Optimization Problems.....		98
Chapter 9 A novel differential crossover strategy based on covariance matrix learning: L-covnSHADE Algorithm		99
9.1.	Introduction	99
9.2.	L-covnSHADE: How it Works	100

9.2.1.	New Crossover Operator.....	100
9.2.2.	Putting it all together.....	102
9.3.	Experimental Results.....	102
9.3.1.	Real-World Benchmark Problems.....	102
9.3.2.	Compared Algorithms.....	103
9.3.3.	Discussion of Results.....	104
9.4.	Conclusion.....	107
Chapter 10 An improved differential evolution using efficient adapted surrogate model: iDEaSm Algorithm		109
10.1	Introduction.....	109
10.2	Surrogate Model.....	111
10.3	iDEaSm Algorithm: How it Works	115
10.3.1	Proposed Adapted Surrogate Model	115
10.3.2	Putting it all together.....	118
10.4	Experimental Results	121
10.4.1	Experiment 1: Bound constrained Benchmark	121
10.4.2	Experiment 2: Real-world Optimization Benchmark	124
10.4.3	Experiment 3: Engineering Design Optimization Problems.....	127
10.4.3.1	Welded Beam Design Problem	127
10.4.3.2	Pressure Vessel Design Problem.....	129
10.5	Conclusion	130
Chapter 11 Conclusions and Recommendations.....		132
11.1	Conclusions.....	132
11.2	Recommendations for Future Work.....	135
AUTHOR'S PUBLICATIONS		137
BIBLIOGRAPHY		140
APPENDIX A: Pseudocodes		158
APPENDIX B: Benchmark Functions		169

APPENDIX C: Algorithms Complexity.....173
APPENDIX D: Experimental Results Tables.....175

List of Figures

Figure 1. 1: Outline of the thesis.....	8
Figure 6. 1: Sinusoidal adjustment configuration	70
Figure 7. 1: An illustration for niching-based population reduction framework	82
Figure 10. 1: Framework for the welded beam design problem [136]	127
Figure 10. 2: Design of pressure vessel problem [137]	129

List of Peseudocodes

Pseudocode A. 1: Pseudocode of canonical DE algorithm.....	158
Pseudocode A. 2: Pseudocode of SFS-DP _{DE-GW} algorithm.....	159
Pseudocode A. 3: Pseudocode of <i>d</i> DSF-EA algorithm	161
Pseudocode A. 4: Pseudo-code of LSHADE-EpSin algorithm	162
Pseudocode A. 5: Pseudo-code of EsDEr-NR algorithm.....	164
Pseudocode A. 6: Pseudocode of L- <i>covn</i> SHADE algorithm.....	165
Pseudocode A. 7: Pseudocode of iDEaSm algorithm	167

List of Tables

Table 3. 1: Comparison of SFS-DP _{DE-GW} with state-of-the-art DE algorithms on $D = 30$ and 50 using the Wilcoxon rank-sum test (significantly, $p < 0.05$).....	45
Table 4. 1: Comparison of d DSF-EA with state-of-the-art algorithms on $D = 50$ and 100 using the Wilcoxon rank-sum test (significantly, $p < 0.05$).....	56
Table 5. 1: Comparison of d DSF-EA with state-of-the-art algorithms on $D = 50$ and 100 using the Wilcoxon rank-sum test (significantly, $p < 0.05$).....	61
Table 5. 2: Average ranking of competitor algorithms, achieved by the Aligned Friedman test at $D=50$. (Best entry is marked in bold).....	63
Table 5. 3: Average ranking of competitor algorithms, achieved by the Aligned Friedman test at $D=100$. (Best entry is marked in bold).....	63
Table 6. 1: Comparison of LSHADE-EpSin with state-of-the-art DE algorithms on $D = 30, 50, 100$ using the Wilcoxon rank-sum test (significantly, $p < 0.05$).....	76
Table 7. 1: Comparison of EsDEr-NR with state-of-the-art algorithms on $D = 50, 100$ using the Wilcoxon rank-sum test (significantly, $p < 0.05$).....	89
Table 8. 1: Average ranking of competitor algorithms, achieved by the Aligned Friedman test at $D=50$. Best entry is marked in bold	96
Table 8. 2: Average ranking of competitor algorithms, achieved by the Aligned Friedman test at $D=100$. Best entry is marked in bold	97
Table 9. 1: Comparison of L-covnSHADE with state-of-the-art algorithms using the Wilcoxon rank-sum test (significantly, $p < 0.05$)	107
Table 10. 1: Average Ranking of competitor algorithms achieved by the	124
Table 10. 2: Average Ranking of competitor algorithms achieved by the	126
Table B. 1: Summary of CEC'14 Test Functions [13]	169
Table B. 2: Summary of CEC'11 real-world numerical optimization problems [112].....	170
Table B. 3: Summary for functions description for bound-constrained benchmark set [11]	171

Table C. 1: Algorithm complexity of SFS-DP _{DE-GW}	173
Table C. 2: Algorithm complexity of <i>d</i> DSF-EA.....	173
Table C. 3: Algorithm complexity of LSHADE-EpSin.....	174
Table C. 4: Algorithm complexity of EsDEr-NR	174
Table D. 1: A comparison of SFS-DP _{DE-GW} and other algorithms on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 30D averaged over 51 runs. Best entries are marked in bold.	176
Table D. 2: A comparison of SFS-DP _{DE-GW} and other algorithms on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.	178
Table D. 3: A comparison of <i>d</i> DSF-EA and other algorithms on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.	180
Table D. 4: A comparison of <i>d</i> DSF-EA and other algorithms on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 100D averaged over 51 runs. Best entries are marked in bold.	182
Table D. 5: A comparison of LSHADE-EpSin and other algorithms on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.	184
Table D. 6: A comparison of EsDEr-NR and other algorithms (first group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.	186
Table D. 7: A comparison of EsDEr-NR and other algorithms (first group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 100D averaged over 51 runs. Best entries are marked in bold.	188
Table D. 8: A comparison of EsDEr-NR and other algorithms (second group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.	190
Table D. 9: A comparison of EsDEr-NR and other algorithms (second group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 100D averaged over 51 runs. Best entries are marked in bold.	192

Table D. 10: A comparison of L-covnSHADE and other algorithms on CEC'11 real-world problems. Mean and standard deviation for each problem. Best entries are marked in bold.	194
Table D. 11: A comparison of iDEaSm and other algorithms. Mean and standard deviation for functions (f_1 - f_{15}) on 10D. Best entries are marked in bold.	196
Table D. 12: A comparison of iDEaSm and other algorithms for bound-constrained benchmark set. Mean and standard deviation for functions (f_1 - f_{15}) on 30D. Best entries are marked in bold.	197
Table D. 13: A comparison of iDEaSm and other algorithms on CEC'11 real-world problems. Mean and standard deviation for each problem. Best entries are marked in bold.	198
Table D. 14: A comparison of iDEaSm and other algorithms for welded beam design problem.....	199
Table D. 15: A comparison of iDEaSm and other algorithms for the pressure vessel design problem.....	199

Abbreviations

Abbreviation	Full Name
EA	Evolutionary Algorithm
GA	Genetic Algorithm
ABC	Artificial Bee Colony
PSO	Particle Swarm Optimization
CA	Cultural Algorithm
DE	Differential Evolution
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
SFS	Stochastic Fractal Search
D	Dimension of the problem
NP	Population Size
g	Generation Number
$rand$	Uniformly distributed random generator in the range [0,1]
$f(\vec{X})$	Objective function
$X_{\min}$	Lower bounds of the problem
$X_{\max}$	Upper bounds of the problem
X	Target vector
V	Mutant vector
U	Trial vector
F	Scaling factor
CR	Crossover rate
FES	Function Evaluations
Ω	Domain of the problem
$mean_{WL}$	Weighted Lehmer mean
$mean_{WA}$	weighted arithmetic mean

Abstract

Engineers and scientists from all disciplines often have to tackle numerous real-world applications. Developing efficient evolutionary algorithms for this target has attracted many researchers due to the fact that many real-world applications can be stated as optimization problems. Differential evolution (DE) has become one of the most effective metaheuristics during the last decade, due to its ability to solve complex optimization problems with diverse characteristics. In this thesis, novel efficient differential evolution variants that can be successfully applied to solve numerical optimization problems are studied. The aim is to develop new improved differential evolution algorithms through mitigating well-known problems that DE suffers from, such as easily getting stuck in local optima, and being easily influenced by the choice of its control parameters. Such improvements should empower these new variants to solve challenging optimization problems efficiently when compared to other existing state-of-the-art algorithms. Different ideas were employed in building such new variants such as: hybridizations that combine the strengths of different canonical algorithms, new ensemble control parameter settings, an improved crossover strategy that is used to build a suitable coordinate system during the search and an assistant surrogate model to mimic the response of the objective function. To validate the performance of the developed algorithms, different challenging test suites from recently developed IEEE-CEC benchmarks were used. Those benchmarks are among the widely used benchmarks by many researchers to test their developed algorithms. Each of them constitutes problems that are tested on different dimensionalities, with a various set of problem features and characteristics, including ruggedness, noise in fitness, multimodality, ill-conditioning, interdependence and non-separability. Moreover, a variety of real-world optimization problems taken from diverse fields are also used. The results of the comparative study statistically affirm the efficiency of the proposed approaches to obtain better results compared to other state-of-the-art algorithms from the literature.

Chapter 1 Introduction

1.1 Motivations and Objectives

In our world, there are countless real-world applications for optimization. Nowadays, many fields of science and companies face many problems in the need of an optimization algorithm or a tool to solve many real-life applications. Many such real-life applications can be formulated as optimization problems because of their nature that one or more objectives need to be optimized. We need optimization in almost every science field such as business, economics and engineering to minimize risk or cost and maximize quality or profit. Indeed, those optimization problems especially with complex landscape pose a challenge for most researchers to solve them in an efficient manner using common mathematical methods, and also evolutionary algorithms.

Differential evolution is considered one of the most efficient evolutionary algorithms for solving different types of optimization problems [1][2][3]. DE was proposed by Storn and Price in 1995 and is an algorithm that is able to utilize several types of mutation and crossover strategies [4]. Four main operations incorporated in DE to simulate the natural evolution process are: initialization, mutation, crossover and selection. Although it has demonstrated its competitive performance compared to other evolutionary algorithms, many theoretical analyses and experimental studies proved that it is still quite dependent on the settings of its control parameters. Such parameters include the scaling factor (F), crossover rate (CR), mutation/crossover strategies and the population size (NP) [5][6]. Early investigations by some researchers of the algorithm suggested a

trial-and-error methodology for tuning its parameters and strategy settings [2] [7]. This method requires tedious trials and hence it is not efficient. Also, DE suffers from premature convergence and stagnation scenarios especially when solving complex optimization problems [8]. Stagnation occurs when the DE algorithm is unable to generate a better offspring and the population has not yet converged, has not found the desired solution, and the diversity is not poor [9]. On the other hand, premature convergence occurs due to loss of diversity when the population converges to local optimum during the search [10].

In this thesis, a potential effort has been made to provide solutions for the aforementioned issues by introducing new improved DE algorithms using synergistic styles and different ways. DE is mainly chosen because of its simplicity, yet powerful, evolutionary search to solve optimization problems. Motivated by these observations, the focus is on improving DE performance to provide solutions for a diverse set of complex optimization problems including real-world problems. IEEE CEC benchmark problems are among the widely used benchmarks by many researchers to test their developed algorithms [11][12][13][14]. Many DE variants have continuously been proposed to successfully solve these problems, and that is an indication of the usefulness of using DE-based algorithms to solve different optimization problems. We consider the use of different CEC benchmark suits on single objective real-parameter numerical optimization to test the developed algorithms. The main motivation is to introduce new improved differential evolution algorithms which are able to produce results that are at least as good as those reported ones in the literature.

1.2 Major Contribution of the Thesis

The main interest is in enhancing the performance of DE algorithm to solve a variety of optimization problems, including real-life problems. To achieve this goal, different novel ideas and techniques are introduced and incorporated with the DE algorithm. The first tactic is to enhance the performance of the DE algorithm by the integration of new and efficient techniques that improve the search algorithm or merge other algorithms or local search methods with DE to form hybrid algorithms. In this regard, two improved hybrid DE algorithms were developed, namely, SFS-DP_{DE-GW} and *d*DSF-EA. The second tactic that is to be introduced in this thesis is to introduce new adaptive methods to control the setting of DE parameters. The adaptive mechanisms can improve DE's performance if designed carefully by updating the parameter settings at each generation. They have shown reliable and faster convergence performance than the deterministic methods or trial-and-error methods. In addition, the adaptive methods are applicable for different optimization problems and real-world applications because they can find the suitable settings for a specific problem based on the feedback of the evolutionary search. For this type of enhancement, two enhanced algorithms were developed as part of this thesis work, namely, LSHADE-EpSin and EsDEr-NR algorithms which use ensemble adaptive control parameter settings.

The third main technique is to introduce an effective surrogate model to assist the DE algorithm to generate more competitive solutions during the search process. The motivation behind using a well-designed surrogate model is to mimic the response of the objective function for the optimization problem being solved.

Hence, more solutions can be generated during the DE evolution without the need to consume the actual pre-defined number of function evaluations (FEs). The developed algorithm, namely iDEaSm, is introduced under this category to solve different types of optimization problems including real-world problems. A new improved crossover strategy to enhance the performance of DE to solve real-world optimization problems is also introduced. The new strategy uses the covariance matrix to establish a suitable coordinate system for the crossover operator. The algorithm is namely *L-covnSHADE*.

To summarize, the major contributions of the thesis are as follows:

- Introduce improved DE algorithms based on hybridization concept with new novel ideas: SFS-DP_{DE-GW} and dDSF-EA algorithms in Chapters 3 and 4, respectively.
- Propose new improved DE algorithms by using ensemble adaptive parameter adaptations based on sinusoidal adjustments: LSHADE-EpSin algorithm in Chapter 6. This algorithm falls in the first class in which an ensemble pool of different adaptations including sine function is used.
- Propose a novel population size adjustment based on niching reduction: EsDEr-NR algorithm in Chapter 7. This is the first algorithm in the literature which uses the Euclidean neighbourhood with the use of niching to adjust the population size.
- Propose a new crossover strategy to enhance the DE algorithm to be able to solve real-world optimization problems: *L-covnSHADE* algorithm in Chapter 9.

- Conduct the use of a well-designed surrogate model to improve the performance of DE when solving diverse set of real-world problems: iDEaSm algorithm in Chapter 10. This is the first algorithm in which a novel adaptation criterion is proposed to adapt the setting of a newly generated surrogate model in each generation.

In this thesis, to verify the performance of the introduced algorithms, I comprehensively used the most recent studies and state-of-the-art algorithms in the comparisons. Different statistical methods are used to judge the performance, and the results affirm the effectiveness of the newly introduced algorithms.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. [Chapter 2](#) elaborates on the canonical differential evolution algorithm and a review of recently developed and state-of-the-art differential evolution algorithms.

The thesis consists of three main parts: improved hybrid differential evolution algorithms, improved differential evolution algorithms based on ensemble parameter adaptation and improved differential evolution algorithms for solving real-world optimization problems.

[Chapter 3](#) and [Chapter 4](#) constitute the first part of the thesis in which two developed algorithms, namely SFS-DP_{DE-GW} and *d*DSF-EA, are elaborated. summary on Part I is presented in [Chapter 5](#). In the second part of the thesis, a new ensemble parameter adaptation is introduced in addition to propose two new

improved DE algorithms, LSHADE-EpSin and EsDEr-NR which are presented in [Chapter 6](#) and [Chapter 7](#), respectively. [Chapter 8](#) presents a summary for Part II with an overall comparison with the recent studies which are proposed to solve the same benchmark suit including the algorithms present in Part I. The third part of the thesis consists of [Chapter 9](#) and [Chapter 10](#) which present improved DE algorithms, *L-covn*SHADE and iDEaSm, respectively, to solve a diverse set of real-world optimization problems. Finally, conclusions and the future scope of the research are summarized in [Chapter 11](#). [Figure 1. 1](#) presents an outline of the thesis.

Figure 1. 1: Outline of the thesis

Chapter 2 Research Background and Literature Review

2.1 Numerical Optimization

Many real-life problems can be formulated as optimization problems in which engineers and scientists have to find the best solution possible. Hence, a considerable amount of research has been devoted which aims to introduce new effective tools and methods to solve such optimization problems. In these problems, the objective is to find a set of decision variables representing the globally optimal solution of an objective function. Without loss of generality, an optimization problem is expressed mathematically as either minimization or maximization of an objective function that is formulated as follows:

$$f(\vec{X}), (f : \Omega \subseteq \mathbb{R}^D \rightarrow \mathbb{R}) \quad (1.1)$$

where Ω is the domain of the problem being solved which involves D decision variables, each of which is represented as a vector $\vec{X} = [x_1, x_2, x_3, \dots, x_D]$ where D is the dimensionality of the problem. As a result, the task of an optimization method is to search for a vector $\vec{X}^*$ that provides the optimal solution for the objective function $f(\vec{X})$ where $f(\vec{X}^*) < f(\vec{X}), \forall \vec{X} \in \Omega$ in case of a minimization problem.

Due to the emergence of solving optimization problems and real-world applications in many science fields, there has been a growing interest in

developing efficient optimization algorithms. Many research works have been proposed in the literature to demonstrate the effectiveness of using evolutionary algorithms and biologically inspired methods instead of using classical methods [15][16][17][18][19][20][21]. These classical methods use differential mathematical calculus to find the optimal solution which requires two main features. Firstly, the objective function of an optimization problem has to be differentiable with respect to the vector $\vec{x}$. Secondly, the derivative has to be continuous. These two requirements limit the use of such methods to solve a class of optimization problems and some real-life applications which may be neither differentiable nor continuous. Also, if these requirements are met, the classical methods such as quasi-Newton [22], Tabu search [23][24], Nelder-Mead's simplex method [25] and non-smooth global optimization methods like sub-gradient method [26] and bundle method [27] can easily get trapped into local optima, and require extra computational cost and time.

In the last few decades, evolutionary algorithms (EAs) became a successful alternative and a powerful tool for the solution of sophisticated problems and has successfully been applied to solve many optimization problems [28][29][30]. Different from classical methods, evolutionary algorithms are population-based techniques which use different operations iteratively to evolve a set of solutions, i.e. individuals instead of a single solution [31]. At least one of these individuals is expected to converge to the best/global optimal solution. Recently, there has been a growing interest in evolutionary algorithms which have been proven with satisfactory solutions in a reasonable time for many optimization problems,

without any prior conditions. Hence, evolutionary algorithms are able to solve any non-differential and/or non-continuous optimization problems.

Pointing at popular evolutionary algorithms, we can refer to genetic algorithms (GA) [32], particle swarm optimization (PSO) [33], ant colony optimization (ACO) [34], artificial bee colony [35], cultural algorithms (CA) [36] and differential evolution (DE) [4]. Despite the fact that these algorithms have been shown as powerful techniques for solving real-valued optimization problems, they can be trapped in a local minimum. Moreover, the global optimum solution cannot be reached when solving complex landscape problems. This thesis mainly investigates the use of differential evolution by developing new improved methods/techniques to solve a diverse set of complex optimization problems and real-life applications in an effective manner.

2.2 Canonical Differential Evolution

DE algorithm is one of the most competitive evolutionary algorithms, simple yet effective, which has proven the quality of its performance to solve a variety of optimization problems [1][2]. This algorithm is originally introduced in 1995 by Storn and Price [4], and later attracted the attention of many researchers to propose new improved state-of-the-art algorithms [1][2]. In DE, four basic steps are required which are: initialization, mutation, crossover and selection.

➤ Initialization

The differential evolution algorithm is a population-based meta-heuristic algorithm which consists of a population of NP individuals where each

individual is expressed as a vector of D -dimensional decision variables as follows:

$$X_{i,g} = (x_{i,g}^1, x_{i,g}^2, \dots, x_{i,g}^D), \quad i = 1, 2, \dots, NP \quad (1.2)$$

where g is the generation number, D is the dimension of the problem being solved and NP is the population size. The algorithm starts initially with a population, denoted as POP_{g_0} where g_0 is the first generation, with randomly distributed individuals $\langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ within the search space of the problem being solved. This initialization for each individual is done as shown below:

$$x_{i,g_0}^j = x_{\min}^j + rand.(x_{\max}^j - x_{\min}^j), \quad j = 1, 2, \dots, D \quad (1.3)$$

where j is the index of parameter value in the i^{th} individual vector at beginning of search denoted as generation g_0 , $rand$ is a uniformly distributed random generator in the range $[0,1]$. $X_{\min} = (x_{\min}^1, x_{\min}^2, \dots, x_{\min}^D)$ and $X_{\max} = (x_{\max}^1, x_{\max}^2, \dots, x_{\max}^D)$ are the lower and upper bounds of each decision variable, x_i^j , of the problem being solved. Then, those initial individuals are evaluated by computing their function values as $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$.

➤ **Mutation operation**

After the initialization step, a new offspring is generated using the mutation operation. DE's mutations are described using the following terminology, DE/ $x/y/z$: x is the solution that is perturbed such as “random” or “best”; y is the number of difference vectors used to perturb x . Each difference vector reflects the difference between two (randomly) selected but distinct population members. z represents the recombination operator used such as binomial (*bin*) or exponential (*exp*).

Five mutation strategies were implemented in the original DE [2] as described below in Eq. (1.4) - (1.8) where $V_{i,g} = (v_{i,g}^1, v_{i,g}^2, \dots, v_{i,g}^D)$ is the mutant vector which is generated against each individual $X_{i,g}$ in the population space. $F_{i,g}$ is the scaling factor that usually takes values within the range [0, 1] and r_1, r_2, r_3, r_4, r_5 are the indices of different randomly selected population members. DE/*best*/1, DE/*current-to-best*/1 and DE/*best*/2 use the best individual in the current population denoted as $X_{best,g}$ to guide the evolutionary search, hence they are called greedy mutations [2].

$$\text{DE}/\text{rand}/1: V_{i,g} = X_{r_1,g} + F_{i,g} \cdot (X_{r_2,g} - X_{r_3,g}) \quad (1.4)$$

$$\text{DE}/\text{best}/1: V_{i,g} = X_{best,g} + F_{i,g} \cdot (X_{r_1,g} - X_{r_2,g}) \quad (1.5)$$

$$\text{DE}/\text{current-to-best}/1: V_{i,g} = X_{i,g} + F_{i,g} \cdot (X_{best,g} - X_{i,g}) + F_{i,g} \cdot (X_{r_1,g} - X_{r_2,g}) \quad (1.6)$$

$$\text{DE}/\text{best}/2: V_{i,g} = X_{best,g} + F_{i,g} \cdot (X_{r_1,g} - X_{r_2,g}) + F_{i,g} \cdot (X_{r_3,g} - X_{r_4,g}) \quad (1.7)$$

$$\text{DE/rand/2: } V_{i,g} = X_{r_1,g} + F_{i,g} \cdot (X_{r_2,g} - X_{r_3,g}) + F_{i,g} \cdot (X_{r_4,g} - X_{r_5,g}) \quad (1.8)$$

➤ Crossover operation

When the mutation phase is completed, the crossover operation is applied to each target vector $X_{i,g}$ and its corresponding mutant vector $V_{i,g}$ to generate a trial vector, $U_{i,g}$ (i.e. offspring). The uniform (binomial) crossover used in the classical DE is given by:

$$u_{i,g}^j = \begin{cases} v_{i,g}^j, & \text{if } (CR < rand) \text{ or } (j = j_{rand}) \\ x_{i,g}^j, & \text{otherwise} \end{cases} \quad (1.9)$$

The crossover rate, CR , is real-valued and is usually specified in the range $[0, 1]$. This variable controls the portion of parameter values that are copied from the mutant vector in this process. j_{rand} is a random integer in the range $[1, D]$. The j^{th} parameter value is copied from the mutant vector $V_{i,g}$ to the corresponding position in the trial vector $U_{i,g}$ if the randomly generated $rand(0,1)$ value is less than or equal to CR . If the condition is not satisfied, then the j^{th} position is copied from the target vector $X_{i,g}$.

The exponential crossover is another commonly used crossover operator as presented in [Eq. \(1.10\)](#). Firstly, integer L is defined which determines the maximum number of components which are copied from $V_{i,g}$ as follows. This condition is valid if a random number, $rand$ is less than or equal to CR , or if L reaches the maximum number which is D :

```

 $L = 0$ 
Do
{
 $L = L + 1$ 
}While ( $(rand < CR)$  and  $(L < D)$ )

```

Then a random position, $n \in [1, D]$ is chosen which acts as a start point where the exchange of components with the mutant vector, $V_{i,g}$ starts. After choosing L and n , the offspring $U_{i,g}$ is obtained as shown below in Eq. (1.10) where $\langle \rangle_D$ denotes the modulo operation with modulus D .

$$u_{i,g}^j = \begin{cases} v_{i,g}^j, & \text{if } j = \langle n \rangle_D, \langle n+1 \rangle_D, \dots, \langle n+L-1 \rangle_D \\ x_{i,g}^j, & \text{otherwise} \end{cases} \quad (1.10)$$

➤ Selection operation

After the final offspring is generated, the selection operation takes place to determine whether the target (the parent, $X_{i,g}$) or the trial (the offspring, $U_{i,g}$) vector survives to the next generation i.e. at $g=g+1$ by comparing the function values. $X_{i,g}$ is replaced by the new offspring $U_{i,g}$ if the latter has a better function value as shown in Eq. (1.11). Otherwise, the new offspring is discarded, and the target vector remains in the population for the next generation.

$$X_{i,g+1} = \begin{cases} U_{i,g}, & \text{if } f(U_{i,g}) \leq f(X_{i,g}) \\ X_{i,g}, & \text{otherwise} \end{cases} \quad (1.11)$$

The above three steps (mutation, crossover and selection) are repeated in each optimization loop to evolve the individuals over the generations, until a termination criterion is met. This criterion is usually defined by a pre-defined number of function evaluations (FEs). A counter is reset at the beginning of the evolutionary search and is incremented each time the objective function is called to evaluate the newly generated individual. When this counter reaches the pre-defined maximum FEs, the optimization process is terminated, and the best-found solution is marked as the solution for the optimization problem being solved. A pseudo-code of the canonical DE is presented in [Pseudocode A. 1](#) in Appendix A.

2.3 Literature Review

Many evolutionary algorithms (EAs) have been developed as population-based algorithms which use operations that are inspired from the natural evolution process to solve a variety of optimization problems [\[15\]\[28\]\[29\]\[30\]](#). One of the major differences that is noticed between the developed evolutionary algorithms is in the reproduction operator [\[15\]](#). This operator defines how the new trial solutions are generated and evolved during the optimization process. Among these algorithms, the differential evolution (DE) is a stochastic metaheuristic algorithm which has shown noticeably good performance in solving a variety of optimization problems [\[4\]\[6\]](#). This algorithm uses: mutation, crossover and selection operations iteratively trying to improve the found solution during the evolutionary search. It is known that DE is dependent on the setting of its control parameters which reflects its performance [\[6\]](#). Those control parameters are: population size (NP), the scaling factor (F), and the crossover rate (CR). In the

early studies, the trial and error method was suggested to control their settings. This method is a time-consuming process and hence is not efficient, and may not lead to good solutions [2]. There are well-studied and general approaches for tuning the control settings such as sequential model-based algorithm control (SMAC) [37] and multi-armed bandit optimization [38].

This consideration motivated many researchers to develop efficient adaptive techniques to control updating the parameters and strategy settings based on different criteria throughout the search. If the parameters and strategy adaptations are well designed, the algorithm's performance can be improved. Many researchers attempted different schemes to control parameter settings for DE such as: deterministic-rules and adaptive/self-adaptive parameter control settings [2]. The first type is a controlled parameter setting according to some deterministic rules without using any information feedback from the optimization search loop. The second scheme is mainly an adaptive control for the parameters based on the interaction between feedback information of the evolutionary search and the control parameters. The key idea is to encode the parameter values within each individual and then evolve them during the search. During this evolution, the values which are capable of generating better offspring are likely to survive to the next generation. This depends on the performance feedback of the used control parameters values during the search. Many newly proposed DE algorithms are classified under this category [1][2]. Other researchers focused on hybridizing the DE algorithm with other evolutionary algorithms or local search methods to enhance the performance of their algorithms. On the other hand, some other approaches have used adaptive control settings and hybridization schemes

[1][2]. There is a plethora of improved DE algorithms in the literature in which some of them have been introduced as improved versions based on previously published ones. The following sub-sections present a review for some of the state-of-the-art DE algorithms. SHADE [39] and L-SHADE [40] algorithms are used as basis algorithms/engines in some of the newly introduced algorithms in this thesis. These two algorithms are improved versions of the JADE algorithm [41]. Hence, these three algorithms have been presented in detail. Also, other improved DE algorithms which proposed in the literature are briefly reviewed.

2.3.1 A Review of some state-of-the-art DE Algorithms

➤ SaDE Algorithm

One of the early DE algorithms that introduces an adaptive scheme to control parameter settings and mutation strategies is SaDE [42]. It implements two mutation types simultaneously: “DE/rand/1” and “DE/current-to-best/1”. It uses the first 50 generations to adapt the probability of generating a new offspring based on one of the two mutation strategies. This adaptation scheme is similar to the mechanism proposed in [43][44][45] where the probabilities of generating an offspring using several heuristics are adapted dynamically. The scaling factor in each generation F_i is adapted using $N(0.5, 0.3)$, a normal distribution with 0.5 and 0.3 for the mean and standard deviation, respectively. The generated values are truncated within the interval (0, 2]. It is believed that this adaptation can provide both local and global values to cover small and large values of F_i that are capable of performing exploration and exploitation. On the contrary, the crossover rate CR_i remains fixed for 5

generations to compute the mean CR_m that will be used in a normal distribution with a standard deviation 0.1 to generate a new CR_i value for the next generation. Initially, CR_m is set to 0.5 and is updated every 25 generations using the past successful CR values from the last update. The quasi-Newton method is used in SaDE to speed up the convergence after 200 generations. Later, SaDE is improved by using a set of five mutation types to solve a set of constrained optimization problems [46].

➤ **jDE Algorithm**

Another algorithm that uses classical DE/rand/1 and adapts F_i and CR_i values that are associated with each individual in a self-adaptive manner is jDE [47]. Different from SaDE algorithm, jDE generates F_i and CR_i with probability equal to 0.1 using a uniform distribution within the intervals $[0.1, 1]$ and $[0, 1]$, respectively. The offspring with the best parameter values are more likely to survive and hence their values are propagated to the next generation. The jDE algorithm is further improved to introduce a new algorithm namely jDE-2 using two mutation types [48]. The results show that this algorithm is competitive to SaDE algorithm when tested on the same benchmark functions.

➤ **dynNP-jDE Algorithm**

dynNP-jDE was introduced to use a novel adaptation of F_i and CR_i with a dynamic population size [49]. It uses the classical rand/1/bin as the mutation strategy. F_i is set to $F_l + rand \cdot F_u$ if a random number is less than t_1 and is kept unchanged otherwise. CR_i is updated using the same scheme if a random number, $rand$ is less than t_2 and is kept unchanged otherwise. $rand$ is a random number that is generated within the interval $[0, 1]$ and t_1, t_2, F_l are set to 0.1 while F_u is set to 0.9. They used a reduction scheme to decrease their large-sized population at each generation.

➤ **EPSDE Algorithm**

In EPSDE algorithm [50], an ensemble of mutation strategies along with a pool of control parameters are used to generate a new offspring. The mutation pool consists of: DE/best/1/bin, DE/best/2/bin, DE/rand-to-best/1/bin, DE/rand-to-best/2/bin, rand/1/bin and rand/2/bin. Distinct schemes are used with different characteristics and mutation strategies. The pool of CR values is chosen from the range $[0.1, 0.9]$ while the parameter F is taken from the range $[0.4, 0.9]$, with a step size equal to 0.1.

➤ **CoDE Algorithm**

In a manner similar to EPSDE, CoDE algorithm is introduced using the idea of incorporating different mutation strategies and parameter settings [51]. It uses three mutation strategies and three control parameter settings. The mutation strategies are: rand/1/bin, rand/2/bin and current-to-rand/1 while it used three fixed values for F and CR .

➤ JADE Algorithm

JADE algorithm is considered one of the most effective and well-known DE algorithms when tested on varying benchmark problems [41]. This algorithm became later a base for many other researchers to build their improved algorithms [39][40][52].

JADE algorithm introduces a new novel mutation strategy namely “*current-to-pbest*” which is an extension of the original “*current-to-best*”. This new mutation uses the top p best individuals to select the best individual denoted as X_{pbest} , to guide the evolution as presented in Eq. (1.12). The reason behind that is to alleviate the stagnation and premature convergence that greedy strategies such as “*current-to-best*” cause over the generations if the best individual, X_{best} , is used in every generation.

$$V_{i,g} = X_{i,g} + F_{i,g} \cdot (X_{pbest,g} - X_{i,g}) + F_{i,g} \cdot (X_{r_1,g} - X_{r_2,g}) \quad (1.12)$$

An optional archive denoted as A is used to store the parents that fail in the selection process, i.e. inferior individuals. Initially, A is filled with a copy of population as first generation. X_{r_2} in Eq. (1.12) is then chosen with a union of current population and the archive A , i.e. $X_{r_2} \in POP_g \cup A$. The motivation behind this is to diversify the population by benefiting from the stored inferior individuals.

After “*current-to-pbest*” is performed, the mutant vectors are checked to be within the predefined search range. For each dimension j , if $v_{i,g}^j$ is outside the

lower and upper boundaries $[x_{\min}^j, x_{\max}^j]$, JADE applies the following correction as shown below:

$$v_{i,g}^j = \begin{cases} (x_{\min}^j + x_{i,g}^j) / 2, & \text{if } v_{i,g}^j < x_{\min}^j \\ (x_{\max}^j + x_{i,g}^j) / 2, & \text{if } v_{i,g}^j > x_{\max}^j \end{cases} \quad (1.13)$$

The Cauchy and normal distributions are used to adapt $F_{i,g}$ and $CR_{i,g}$ for each individual with means equal to μF_g and μCR_g , respectively, and a value of 0.1 is used for the standard deviation as shown below:

$$F_{i,g} = \text{Cauchy}(\mu F_g, 0.1) \quad (1.14)$$

$$CR_{i,g} = \text{Normal}(\mu CR_g, 0.1) \quad (1.15)$$

Initially, the two mean values are set to 0.5. Later, they are updated at the end of each generation using the previously stored successful values of $F_{i,g}$ and $CR_{i,g}$ as shown in Eq. (1.16) and Eq. (1.17), respectively. $F_{i,g}$ and $CR_{i,g}$ are considered successful values if the newly generated offspring is better than its target vector/parent. Those values are stored in two sets denoted as S_F and S_{CR} respectively.

$$\mu F_{g+1} = (1 - c) \cdot \mu F_g + c \cdot \text{mean}_L(S_F) \quad (1.16)$$

$$\mu CR_{g+1} = (1 - c) \cdot \mu CR_g + c \cdot \text{mean}(S_{CR}) \quad (1.17)$$

where mean_L is the Lehmer mean which is computed as shown below:

$$mean_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F} \quad (1.18)$$

➤ SHADE Algorithm

SHADE is a new extension of JADE algorithm [39] which stands for success history adaptive DE. The authors of SHADE have two main additions to improve the performance which are: dynamically changing the greediness factor in “*current-to-pbest*” and using a history-based parameter adaptation. Instead of using a fixed value for the greediness factor denoted as parameter p , SHADE dynamically adjusts this parameter for each individual as shown below where $p_{\min}$ is set to $2 / NP$:

$$p_i = rand [p_{\min}, 0.2] \quad (1.19)$$

Different from JADE, a memory of H entries is used to store the two mean values: μF_g and μCR_g . At the beginning of the search, this memory is initialized with 0.5 for both means. Cauchy and normal distributions are still used as shown in Eq. (1.16) and Eq. (1.17) to generate $F_{i,g}$ and $CR_{i,g}$ except that a random index, $r_i \in [1, H]$ is used to select the mean values μF_g and μCR_g from memory. The external archive is also used in SHADE in a manner similar to JADE.

The memory is updated at the end of each generation. The index k determines the position to be updated and is incremented when new values enter the

memory. The two means μF and μCR at position k are updated as shown below:

$$\mu F_k = mean_{WL}(S_F) \quad (1.20)$$

$$\mu CR_k = mean_{WA}(S_{CR}) \quad (1.21)$$

Where $mean_{WL}$ is the weighted Lehmer mean and $mean_{WA}$ is weighted arithmetic mean and which are computed as shown below. The weight w_k is defined by the difference in function values of the offspring and its parent denoted as Δf_k divided by the total sum of the difference in values of every other successful individual. Later, different new extended versions of SHADE are introduced in the literature [53][54].

$$mean_{WL}(S) = \frac{\sum_{k=1}^{|S|} w_k \cdot S_k^2}{\sum_{k=1}^{|S|} w_k \cdot S_k} \quad (1.22)$$

$$mean_{WA}(S) = \sum_{k=1}^{|S|} w_k \cdot S_k \quad (1.23)$$

$$w_k = \frac{\Delta f_k}{\sum_{j=1}^{|S|} \Delta f_j} \quad (1.24)$$

$$\Delta f_k = |f(U_{k,g}) - f(X_{k,g})| \quad (1.25)$$

➤ L-SHADE Algorithm

L-SHADE algorithm uses the same engine of SHADE algorithm with two modifications [40]. The first one is the use of weighted Lehmer mean to update the two means μF and μCR at position k of memory, instead of using the usual arithmetic mean to update μCR as in SHADE. Hence, the following equation is used:

$$\mu CR_k = \text{mean}_{WL}(S_{CR}) \quad (1.26)$$

Different from SHADE which uses a fixed population size over the generation, i.e. $NP = 100$, L-SHADE uses a linear population size reduction to adjust the population size at each generation, g . At the beginning of the search, a large population size is used denoted as $NP_{\max}$, and then the following equation is used to dynamically decrease the size for the next generation:

$$NP_{g+1} = \text{Round} \left[\left(\frac{NP_{\min} - NP_{\max}}{FEs_{\max}} \right) \cdot FEs + NP_{\max} \right] \quad (1.27)$$

where $NP_{\min}$ is set to 4 which is the minimum number of individuals needed to perform “*current/to/pbest*” mutation. $NP_{\max}$ is the maximum initial size of the population and is set to $18 \times D$ where D is the dimensionality of the problem being solved. FEs is the current function evaluations and $FEs_{\max}$ is the predefined maximum number of function evaluations.

L-SHADE algorithm was ranked the winner of the single objective real parameter competition of the CEC’14 on real parameter single objective

optimization [55]. The same authors of L-SHADE introduced two new extended variants namely R-SHADE and R-LSHADE in [56] where they utilize different restart variants much like restart CMA-ES variants [57][58][59]. In R-LSHADE, the restart strategy is responsible for resetting the search. It does this when the search is not able to find any better solution at later generations. Different from R-SHADE, RL-SHADE is reset when the population size reaches the minimum value. It is worth noticing that this restart strategy resets the whole search and is not responsible for doing any reset for the individuals within a single run.

Later, L-SHADE algorithm attracted the attention of many researchers to use its engine and introduce new improved and extended versions [60][61][62][63][64][65][66][67]. In [68], L-SHADE is incorporated with a successful-parent selection scheme and an eigenvector-based crossover. This algorithm namely, SPS-LSHADE-EIG, was tested on CEC 2015 benchmark set and showed competitive performance.

2.3.2 Other Improved DE Algorithms

Over the last few years, different improved DE algorithms have been proposed in the literature. In 2015, Yang *et al.* [69] proposed an auto enhanced population diversity at the dimension level. The algorithm measures the distribution of the population at each dimension to identify stagnation cases. If stagnation is caught during the search, the population is diversified to an appropriate level and this enables the algorithm to get a better convergence rate. Li *et al.* [70] used two models to improve the performance of DE. The first model is a distributed model

which is used to generate new better individuals and increase the explorative ability. The other model is a centralized model in which they used an evolution path based on covariance matrix adaptation (CMA) to increase the convergence speed. The authors in [71] firstly introduced the successful parent selection in DE. In their method, if an individual is stagnated for a consecutive number of generations, the parents are chosen from an external archive which stores the previous successful solutions. An adaptive ranking mutation operator is introduced in DE in which the individuals are ranked based on three situations: infeasible, feasible and semi-feasible [72]. This method is proposed to solve constrained optimization problems. The authors in [73] used an iteration temporal randomness to self-adapt the control parameters in DE. They incorporate a chaotic behaviour as a random level parameter to select the suitable settings and to improve the individuals' contribution during the elitist selection.

Arithmetic recombination is incorporated with DE with an ensemble of parameters for solving multi-modal optimization problems [74]. In this algorithm, the authors aim at enhancing the exploration and improve the exploitation by arithmetic recombination on the trail vector using three random individuals with neighborhood mutation. Very recently, the authors in [75] proposed a new differential evolution algorithm based on a two-step sub-population strategy. In this algorithm, the population is divided into two sub-populations: inferior and superior sub-populations. The inferior sub-population is responsible for generating a new offspring. If the new offspring is better than the target vector in the superior population, it will replace the latter and the individuals in inferior sub-population are used as vectors for mutation. The

algorithm is applied on single and multi-objective problems and it has also been demonstrated to solve the microwave circuit design problem.

In [76], the authors incorporated the idea of multi-mutation scheme in differential evolution. In this algorithm, the mutation strategy generates multiple mutant vectors instead of one, and the final mutant vector is generated by combining the information of those multiple mutant vectors. An adaptive multi-population differential evolution with dynamic population reduction is proposed by our group in [77]. In this algorithm, the population is clustered into multiple tribes and utilizes an ensemble of different mutation and crossover strategies. A competitive success-based scheme is introduced to determine the life cycle of each tribe and its participation ratio for the next generation. In each tribe, a different adaptive scheme is used to control the scaling factor and crossover rate.

Recently in 2017, an improved DE algorithm is introduced in the literature to solve constrained experimental domains for uniform designs based on a two-phase scheme [78]. In the first phase, a clustering method is used to guide the DE population for different search directions, and to obtain different good samples for experimental domain. Then the goal during the second phase is to scatter these collected samples in the experimental domain uniformly by maximizing the minimum Euclidean distance between them. In [79], a new enhanced DE algorithm is proposed to solve big data optimization problems. The idea is mainly based on splitting the big problem into smaller sub-problems using the cooperative co-evolution method. Then, these smaller sub-problems are solved using different DE operators.

Very recently in 2018, the canonical DE algorithm is further enhanced by using multiple local search methods to solve constrained optimization problems [80]. The coordination between these local search methods is done through a probabilistic method. This method is responsible for activating the most suitable local search based on a cost benefit scheme. Another enhanced DE algorithm which proposes the use of a generalized scaling factor is proposed in [81]. In this algorithm, the covariance matrices are used to transform the scaling factor, and the mutant operators of “DE/*rand*/1” and “DE/*best*/1”. The reason behind that is to obtain synthetic mutation operators with a general parameter tuning. A detailed review for the newly introduced L-SHADE, SHADE and JADE extended variants is introduced in [82]. The authors aim to study these algorithms with a deep insight on their performances to solve real-world optimization problems. Also, to provide a comparative study which points out the conditions in which an algorithm can be considered to be successful or unsuccessful to solve specific problems. Another detailed review and taxonomy is introduced in [83], which provides a detailed study on the adaptive DE control settings and their algorithmic design issues.

2.4 Summary

In this chapter, the numerical optimization is briefly defined with a general mathematical explanation for any optimization problem. It highlighted the importance of optimization in almost every science field, hence the need to develop efficient optimization algorithms as they attracted the attention of many researchers. In recent years, evolutionary algorithms are used to solve a wide spectrum of optimization problems. Those algorithms are meta-heuristic

population-based techniques which use different mutation and crossover operators iteratively to solve an optimization problem. Differential evolution is one of the most effective and powerful evolutionary algorithms which shows competitive performance in solving many optimization problems.

The canonical differential evolution is comprehensively reviewed with a detailed explanation of how it works. Although its effectiveness and competitive performance has been proved, many experimental studies and theoretical analysis show that it is sensitive and still quite dependent on the settings of control parameters. Also, it fails to solve complex optimization problems effectively which is caused by stagnation and premature convergence. Several improved algorithms have been introduced to solve these issues.

It has been shown in the literature review that there is a plentiful of improved differential evolution algorithm that were developed in the last era. These algorithms use different techniques, self-adaptive schemes and synthetic methods. Although these methods succeed in enhancing the performance of DE when solving a diverse set of optimization problems, locating the optimal solution is still a challenging task in most of these methods for complex optimization problems. Finding this solution with the specified number of function evaluations state a challenge for most researchers. Furthermore, it still remains a need to advance the performance of DE and existing methods especially with the developing of new optimization problems and contemporary real-life applications.

These considerations offer a clear direction for further investigations of our research in the following chapters. The goal is to develop new and novel ideas/methods to enhance the performance of DE to solve the recently developed benchmark suits and real-life applications. In our research, we focus mainly on developing new efficient DE algorithm which is able to produce results that are at least as good as the best reported ones in the literature.

PART I

Improved Hybrid Differential Evolution

Algorithms

Chapter 3 Hybrid differential evolution and stochastic fractal search: The SFS-DP_{DE-GW} algorithm

3.1 Introduction

The aim of hybridization in the context of evolutionary computation is to combine appropriate operators from different evolutionary computation paradigms to form a single technique that enjoys a statistically superior performance over a wide range of optimization problems. One of the well-known evolutionary algorithms that embraces the concept of hybridization is AMALGAM-SO [84]. AMALGAM-SO hybridizes three different algorithms which are PSO, GA and CMA to develop a new powerful algorithm. A self-adaptive learning scheme is used to determine the participation ratio of individuals that each algorithm can use to generate a new offspring in each generation of the search process. This algorithm showed promising results on solving high dimensional complex multimodal problems. UMOEAs algorithm is another successful example that hybridizes three multi-operator evolutionary algorithms which are: GA, DE and covariance matrix adaptation evolution strategy (CMA-ES) [85]. The algorithm consists of three sub-populations and each sub-population is evolved using one of the aforementioned algorithms. The success rate of each of the used algorithms is recorded and the best algorithm is used to evolve its own sub-population while other sub-populations are kept on

hold. Examples of other successful hybridization of DE algorithms that provides a competitive performance can be found in [2].

Although the above examples show the advantages of hybridization style when attempting to solve complex problems, it also comes with a potential cost. Firstly, when more than one algorithm is used, the search engine requires more computational resources to organize the work between different algorithms and to work in an effective manner. Secondly, an intermediate coordinate system is needed when more component algorithms are included to balance between exploration, exploitation and the participation ratio of such individual algorithms at each generation. It is important thus to develop a hybrid algorithm which is simple without extra computation resources, and also works in an effective manner compared to constituting component algorithms.

This chapter introduces a novel hybridization between DE and stochastic fractal search algorithm (SFS). Recently, SFS was introduced as a new powerful metaheuristic algorithm [86]. This algorithm imitates the natural phenomenon of growth and uses two main processes which are: the diffusion process and the update process. The diffusion process uses Gaussian walks in which each particle is diffused around its current position, and hence this process is considered the exploitation phase. After that, two update processes are called in an attempt to increase the search ability and discover new search directions, i.e. empower the exploration phase. Although SFS overcomes the shortcomings of its predecessor nature-inspired algorithm which is fractal search, a competitive performance is not guaranteed when solving complex optimization problems due to stagnation and premature convergence scenarios. We aim at mitigating such issues and

introduce an enhanced and powerful evolutionary algorithm that hybridizes DE and SFS algorithms to form a new algorithm namely SFS-DP_{DE-GW}. Instead of just using the Gaussian walks in the diffusion process that may suffer from low convergence, DE operators based on SHADE mutation is also used. The new adjustment of the diffusion process increases the search ability of the new amalgam and balances the exploitation and exploration phases. The strength of the update process is also enhanced by providing fitter search agents to start the exploration phase.

The remaining sections complete this chapter as follows. Section 3.2 briefly introduces the SFS algorithm. In Section 3.3, the proposed method is elaborated. Section 3.4 describes the used benchmark problems, parameter settings and simulation results with a comparative study against other state-of-the-art algorithms. Finally, Section 3.5 summarizes the conclusion of this chapter.

3.2 Stochastic Fractal Search

Stochastic fractal search is a new metaheuristic algorithm that was introduced in 2015 by Salimi as an improved version of the basic fractal search and to overcome its shortcomings [86]. The basic fractal search has many parameters that need to be tuned and designed well to reflect good results. Besides, there is no information exchange between particles. This affects the convergence rate and makes the search performs independently for all particles. Therefore, stochastic fractal search was introduced to tackle these shortcomings.

SFS consists of two main processes: the diffusion process and the update process. The diffusion process simulates the basic fractal search by satisfying the

intensification or exploitation property in which each particle is diffused around its current position. This property is essential because it increases the chance of locating global optima and prevents the scenario of getting trapped in a local optimum. The diffusion process is different from the basic one in fractal search as it is considered a static process and does not cause a dramatic increase in the number of participating points to generate a new particle. In other words, the best generated particle from the diffusion process is considered and other remaining ones are discarded. On the other hand, the update process is considered as a diversification or an exploration phase in which each particle updates its position based on the positions of the other selected particles in the population.

Initially, the search starts by initializing a set of particles $\langle X_{1,g}, X_{2,g}, \dots, X_{NP,g} \rangle$ uniformly distributed within the search range of the problem being solved as shown earlier in [Eq. \(1.3\)](#), where NP is the number of particles. Next, these initial particles are evaluated by computing their function values. After that, each particle is diffused, and a new particle denoted as $X_{i,new}$ is generated for each X_i using one of the following Gaussian walks as show below.

$$X_{i,new} = \text{Gaussian}(\mu_{BP}, \sigma) + (rand \times X_{best} - rand \times X_i) \quad (2.1)$$

$$X_{i,new} = \text{Gaussian}(\mu_p, \sigma) \quad (2.2)$$

Where *Gaussian* is a Gaussian walk generator with a mean μ and a standard deviation σ , *rand* is a uniform random number in the range $[0, 1]$, X_{best} is the best particle in the population. The two means μ_{BP} and μ_p are set to X_{best} and X_i

respectively while σ is computed as shown in Eq. (2.3) where g is the generation number.

$$\sigma = \left| \frac{\log(g)}{g} \times (X_i - X_{best}) \right| \quad (2.3)$$

Instead of using Levy flight as in the basic fractal search, the diffusion process in SFS uses the Gaussian walk as it is better in locating the global optimum since Levy flight converges faster. SFS allows the points to be diffused more than once, and this is determined using a predefined number denoted as γ which specifies the diffusion times. After the diffusion process is performed, the newly generated particles are evaluated and the best NP particles, according to their function values, are selected for next process.

After that, the update process is started which is considered a diversification or an exploration phase in which each particle updates its position based on the positions of the other selected particles. This process consists of two update procedures. Firstly, each particle is given a rank Pa_i according to Eq.(2.4), where $rank(X_i)$ is the rank of each particle based on its function value.

$$Pa_i = \frac{rank(X_i)}{NP} \quad (2.4)$$

The first update process is applied for each index vector in every selected particle according to its rank. If a random number, $rand$ is less than Pa_i , Eq. (2.5) is used to update the j^{th} index of the selected particle, i.e. X_i^j . Otherwise it remains unchanged.

$$X_i^j = X_{r_1}^j - rand \times (X_{r_2}^j - X_i^j) \quad (2.5)$$

Where X_{r_1} and X_{r_2} are two different random particles which are chosen from the population. After performing the above update process for every index in X_i , the new generated particle replaces the old if it has a better function value.

The second update process uses two different equations to update the position of every particle in the population. Once again, all the points obtained from the first update process are ranked using Eq. (2.5). Next, if $rand$ is less than Pa_i , Eq. (2.6) is used to update the current particle X_i . Otherwise, no update process takes place. The first equation is applied if a random number $rand$ is less than or equal to 0.5 which uses the best individual X_{best} and a random individual X_{r_1} . Otherwise, the current position of X_i is updated using two random individuals denoted as X_{r_1} and X_{r_2} . Similar to the first update process, if the function value of the new particle is better than the old one, the replacement occurs.

$$X_i = \begin{cases} X_i - rand \times (X_{r_1} - X_{best}), & \text{if } rand \leq 0.5 \\ X_i + rand \times (X_{r_1} - X_{r_2}), & \text{otherwise} \end{cases} \quad (2.6)$$

3.3 SFS-DP_{DE-GW}: How it Works

3.3.1 Differential Evolution Diffusion Process

Instead of only using Gaussian walks in the original stochastic fractal search, the diffusion process is also carried out via differential evolution. SFS-DP_{DE-GW} algorithm, as part of the production for this PhD work, used SHADE mutation [39] in the diffusion process besides the Gaussian walks. In other words, the proposed algorithm hybridizes DE operators and Gaussian walks in the diffusion process of SFS. Hence, the proposed diffusion process is named: differential evolution diffusion process. This provides an ensemble pool consisting of two schemes to guide the evolutionary search of the new hybrid algorithm namely SFS-DP_{DE-GW}. In this algorithm, the selection of those schemes is done randomly. The proposed diffusion process starts by generating a random number *rand* between 0 and 1. If the generated number is less than 0.5, the two Gaussian walks of the original SFS is called as presented in Eq. (2.1) and (2.2). Otherwise, SHADE mutation, as explained earlier in subsection 2.3.1, is called to generate new better individuals.

SHADE mutation is mainly chosen due to its efficiency that enhances the “current-to-*p*best/1” mutation strategy which is originally developed by JADE algorithm [41]. Besides that, SHADE introduced effective adaptive settings for *F* and *CR* parameters using a history-based scheme. This increases the diversity of the population and allows information exchange by propagating the best knowledge/parameter values to generate a new better offspring. Also, merging the SHADE mutation in the diffusion process helps boosting the Gaussian walks

by hybridizing the DE operates to generate new better particles. As recommended by SFS authors [86], the first Gaussian walk (Eq. (2.1)) was usually used for simple problems and the second one (Eq. (2.2)) walk was usually used for hard problems. In the proposed algorithm and after experimentation, both of them were used to benefit from their characteristics to solve a diverse set of optimization problems. The selection between those two Gaussian walks is done randomly based on a random walk number, w which is generated between 0 and 1. If w is less than 0.5, Eq. (2.1) is called. Otherwise, Eq. (2.2) is used.

3.3.2 Putting it all together

A full pseudocode of SFS-DP_{DE-GW} is presented in Pseudocode A. 2 of Appendix A. The algorithm starts with the initialization step as presented in Lines 1-6. In this step, the initial control parameter settings are set and a population of NP D -dimensional individuals is initialized as shown earlier in Eq. (1.3). Next, the evolutionary search is started which hybridizes the DE and SFS operators to generate new better individuals for each generation, g . The DE diffusion process is performed which calls the SHADE mutation or Gaussian walk randomly to generate a new set of offspring individuals. Lines 11-24 describe how the new individuals are generated if the Gaussian walk scheme is chosen. For every individual in the population, $X_{i,g}$, a new individual/offspring is generated based on a randomly chosen Gaussian walk (Eq. (2.1) or Eq. (2.2)). Next, the new offspring is evaluated, and the replacement occurs if it has better function value compared to its parent as presented in Lines 19-23.

Lines 26-42 describes SHADE operators which starts by generating $F_{i,g}$ and $CR_{i,g}$ based on Cauchy and normal distributions as presented in Eq. (1.14) and (1.15) respectively. Then, the “current-to- p best” mutation strategy with binomial crossover is performed to generate a new offspring. In case the new generated individual is better than its parent according to the computed function values, the replacement occurs and the successful $F_{i,g}$ and $CR_{i,g}$ are stored in S_F and S_{CR} , respectively. Those values are used at the end of the optimization loop to update the memory for next generation as presented in Line 40.

After that, the update process of the original SFS starts as presented in Lines 43-66 to enhance the quality of the generated population using the differential evolution diffusion process. The first update process is presented in Lines 45-56 in which Eq. (2.5) is used to update each index of $X_{i,g}$ if the condition in line 48 is satisfied. Lines 57-66 presents the second update process which uses Eq. (2.6) to update $X_{i,g}$ according to its rank denoted as Pa_i . For the two update processes, the new individual is evaluated, and the replacement occurs if it has better function value.

3.4 Experimental Results

3.4.1 Numerical Benchmark

The algorithm is tested on a set of 30 benchmark optimization problems from the special session and competition on real-parameter optimization held under the IEEE CEC’2014 benchmark problems [13]. This benchmark suite spans a various

set of problem features and characteristics, including ruggedness, noise in fitness, multimodality, ill-conditioning, interdependence and non-separability. These benchmarks are based on classical functions such as Rastrigin's, Schwefel's, Rosenbrink's, Ackley's, and Griewank's functions. To summarize, functions f_1 – f_3 are unimodal, functions f_4 – f_{16} are multimodal, functions f_{17} – f_{22} are hybrid, and functions f_{23} – f_{30} are composition functions. [Table B. 1](#) of Appendix B presents a summary for these functions where F_i^* is the optimum for function i . The search range of these functions is $[-100,100]$. As stated in the technical report, each algorithm should run 51 times for each test problem with function evaluations equals to $10,000 \times D$ where D is the dimension of the problem and can be tested on values equal to 10, 30, 50 and 100 dimensions. The error is the absolute value of the change between the real optimum value of the objective function, $F_i^* = F_i(X^*)$ and the best result $f(X_{best})$ found by an algorithm, i.e., $|F_i^* - f(X_{best})|$. When the difference between the best solution found and optimal is equal to or less than 10^{-8} , the error is treated as 0. More details on these functions can be found in [\[13\]](#).

3.4.2 Parametric Setup and Algorithm Complexity

The parameter values for SFS-DP_{DE-GW} are set as follows:

- Population size (NP) is set to 100
- For SHADE mutation: memory size (H) is set to 10 and the initial mean values for μF and μCR are set to 0.5
- For diffusion process: maximum number of diffusion, γ is set to 1

The algorithm was coded using Matlab 2014 and was run on an Intel® Core™ i7 CPU at 3.4 GHz speed with 8 GB RAM. The algorithm complexity is computed following the guidelines of CEC'14 benchmark problems. More details can be found in [Appendix C](#). [Table C. 1](#) presents the complexity of SFS-DP_{DE-GW} which is reflected by $T_1, \hat{T}_2$ and $(\hat{T}_2 - T_1)/T_0$ as inspired by competition rules.

3.4.3 Compared Algorithms

The performance of SFS-DV_{DE-GW} algorithm was compared with the following state-of-the-art algorithms:

1. CoDE: differential evolution with composite trial vector generation strategies and control parameters [\[51\]](#)
2. EPSDE: Differential evolution algorithm with ensemble of parameters and mutation strategies [\[50\]](#)
3. JADE: Adaptive differential evolution with optional external archive [\[41\]](#)
4. SaDE: Self-adaptive differential evolution algorithm [\[42\]](#)
5. SHADE: Success-history based parameter adaptation for differential evolution [\[39\]](#)
6. SFS: Stochastic fractal search [\[86\]](#)

The same control parameter values that were suggested in the original papers were used to run the codes. Following the guidelines of the used benchmark set as explained in [\[13\]](#), each algorithm is run for $10,000 \times D$ functions evaluations for 51 independent runs. For SFS algorithm, the same control parameters as recommended by the authors, were used: the maximum diffusion number (γ) is set to 1 and the population size is set to 100.

3.4.4 Results on Numerical Benchmark

The 30 benchmark problems are tested on $30D$, and $50D$. A comparative study using the mean and standard deviation of the best-of-run errors for 51 independent runs for all the used algorithms on $30D$ and $50D$ is given in [Table D. 1](#) and [Table D. 2](#), respectively. The best mean error values are marked in bold. A two-sided Wilcoxon rank sum test [87] is used to evaluate the statistical significance of the observed differences between SFS-DP_{DE-GW} and every contestant algorithm at the 0.05 significance level. As indicated in the results tables, a “–” sign signifies the case where the compared algorithm shows an inferior performance, and a “+” sign identifies when it exhibits superior performance. In such cases, H_0 is rejected. When the performance difference is not statistically significant, these cases are marked with a “=” sign. Moreover, according to the obtained Wilcoxon’s test, the results are summarized as “ $w/t/l$ ” in [Table 3. 1](#) and also the last row of [Table D. 1](#) and [Table D. 2](#), to imply that the contestant algorithm wins in w functions, ties in t functions, and loses in l functions, in a pairwise comparison with the proposed algorithm.

Referring to these tables, SFS-DP_{DE-GW} is at least as good as the other algorithms in 21 functions on both $30D$ and $50D$. The performance of SFS-DP_{DE-GW} is not degraded when increasing the dimensionality to $50D$. This affirms the effectiveness and robustness of the proposed algorithm. The summary of Wilcoxon test as can be seen in [Table 3. 1](#) shows that there is a clear difference between the proposed algorithm and its used components which are: SFS and SHADE algorithms. Compared to SHADE algorithm, SFS-DP_{DE-GW} loses in just 2 functions at $30D$ and $50D$. Also, the number of equal performances has been

decreased from 16 to 14, while the number of times it lost has been increased from 12 to 14 for 30D and 50D respectively. SFS wins in 2 out of 30 functions on 30D and 50D compared to SFS-DP_{DE-GW}, while it shows equal performances in 8 functions on both dimensions and loses in 20 and 19 functions on 30D and 50D respectively. This demonstrates the effectiveness of the proposed differential evolution diffusion process which hybridizes SHADE with a mixture of two Gaussian walks.

Table 3. 1: Comparison of SFS-DP_{DE-GW} with state-of-the-art DE algorithms on $D = 30$ and 50 using the Wilcoxon rank-sum test (significantly, $p < 0.05$)

Vs. SFS-DP _{DE-GW}		$D = 30$	$D = 50$
SaDE	+ (win)	0	0
	= (equal)	6	4
	- (lose)	24	26
CoDE	+ (win)	6	2
	= (equal)	13	8
	- (lose)	11	20
EPSDE	+ (win)	3	3
	= (equal)	9	6
	- (lose)	18	21
JADE	+ (win)	1	0
	= (equal)	15	13
	- (lose)	14	17
SHADE	+ (win)	2	2
	= (equal)	16	14
	- (lose)	12	14
SFS	+ (win)	2	3
	= (equal)	8	8
	- (lose)	20	19

3.5 Conclusion

In this chapter, a new hybrid differential evolution algorithm is proposed that combines the explorative and exploitative capabilities of: stochastic fractal search and SHADE operators. The proposed algorithm namely, $\text{SFS-DP}_{\text{DE-GW}}$ embeds differential evolution in the diffusion process of stochastic fractal search. In this algorithm, a new adjustment to the diffusion process of stochastic fractal search was introduced by hybridizing the SHADE mutation and a mixture of two Gaussian walks. This combination enhances the evolutionary search clearly and increases the efficiency to produce high quality solutions. The performance of the algorithm is evaluated on a set of 30D optimization problems of CEC'14 benchmark problems. The results show that $\text{SFS-DP}_{\text{DE-GW}}$ has a competitive performance and a favourable scalability behaviour compared to other state-of-the-art algorithms.

Chapter 4 A Decremental Stochastic Fractal

Differential Evolution: The *d*DSF-EA algorithm

4.1 Introduction

In this chapter, a new extension of the previously presented algorithm in Chapter 3, SFS-DP_{DE-GW}, is presented. The coherency between the differential evolution and stochastic fractal search using the nature of exploitative and explorative powers helps to develop a new successful optimizer to guide search towards more successful solutions. The individuals can then evolve in a better manner instead of merely generating new individuals and waste extra computations without showing good performance. A new success-based scheme is used to utilize the update processes and to solve the burden of extra computations during the search. This new algorithm captures the strengths of both component algorithms and produces a greater explorative power as compared to the original components.

The new proposed hybrid algorithm, namely decremental differential stochastic fractal evolutionary algorithm, *d*DSF-EA presents a three-stage optimization algorithm: differential evolution diffusion process, success-based update process and population size dynamic reduction. In the first stage, we use the same engine of SFS-DP_{DE-GW} which hybridizes SHADE and two Gaussian walks to generate new offspring at each generation. In the second stage, an adaptive success-based mechanism to call the update processes is introduced. In the original SFS algorithm, the two update processes use a fixed number of function evaluations at each generation. Hence, if the update process is unable to generate good

solutions, few function evaluations may be lost. The use of success-based scheme of the update processes is proposed to solve this issue. The proposed success-based scheme will assign a suitable number of function evolutions for each update process based on its success in previous generations. Hence, function evaluations will be utilized in an effective manner. A linear population size reduction is used in the last stage after the update process is performed to improve the whole performance of the proposed amalgam. Population size reduction has been shown to be highly effective in improving evolutionary algorithm's performance [49][88][40]. Using those three stages, the proposed algorithm has a guided synergy to efficiently select appropriate solution directions by balancing exploration and exploitation during the search.

The rest of the chapter is organized as follows. In Section 4.2, the new algorithm, *dDSF-EA*, is elaborated with a detailed discussion. Simulation results are presented in Section 4.3 with a comparison of *dDSF-EA* with other state-of-the-art algorithms. Finally, concluding remarks are summarized in section 4.4.

4.2 *dDSF-EA*: How it Works

The update process in stochastic fractal search is considered as the exploration phase while the diffusion process is considered as the exploitation phase. However, there is no balance between these two phases in terms of used function evaluations (FEs). The number of function evaluations that are needed by the diffusion process equals to $\gamma \times NP$, while the update process for both procedures takes $2 \times NP$ where γ is the maximum diffusion number and NP is the population size. This means that in each generation, SFS algorithm needs at least

$3 \times NP$ function evaluations to generate NP new individuals if γ is set to 1. As a result, if the population cannot maintain a balance between exploration and exploitation search phases, SFS algorithm may suffer from stagnation and premature convergence scenarios. The same issue is reflected in SFS-DP_{DE-GW} algorithm. Motivated by these observations, the proposed algorithm in this work uses the basic engine of SFS-DP_{DE-GW} and adds a new success-based scheme for the update process to provide a balance between exploration and exploitation phases. This provides a dynamic strategy for calculating the participation ratio that each update process can use and saves extra computations instead of consuming extra computations without any benefit. Furthermore, a population size reduction is used after the update process is called. Hence, d DSF-EA algorithm presents a three-stage optimization algorithm. These stages are as follows:

- Differential evolution diffusion process
- Success-based update process
- Population size dynamic reduction

The first stage, differential evolution diffusion process, is presented in SFS-DP_{DE-GW} as explained earlier Section 3.3.1. The following subsections explain the other two main phases in details.

4.2.1 Success-based Update Process

The update process simulates how each individual updates its position based on the position of other points in the population. One of the shortcomings that SFS suffers from is that the update phase consumes at least double the computations that the diffusion process takes if maximum diffusion, γ is set to 1. On the other hand, the diffusion process consumes at least double the computations of update process if $\gamma \geq 2$. In other words, there is no balance between exploration (update process) and exploitation (diffusion process). To tackle this issue, a success-based scheme for the update process is introduced. According to this scheme, the update process is considered a dynamic process in which the participation ratio denoted as Φ (i.e. the number of individuals assigned to each update process) is dynamically updated in each generation. At first generation, Φ_i for each of the two update processes is set to $\frac{NP}{2}$ different from SFS and the proposed algorithm SFS-DP_{DE-GW} where both use NP . Later for other generations, Φ_i is determined by the mean success rate for an update process from the previous generations as shown in [Eq. \(2.7\)](#).

$$\Phi_{i,g} = \lfloor \text{mean}(SR_{i,g}) \rfloor \quad (2.7)$$

Where i can be 1 to denote the participation ratio for first update process, and 2 for second update process. SR_g is the success rate of an update process at generation g . The newly generated individual by an update process is marked successful if it has a better function value compared to its old corresponding without update. If this happens, the success rate of an update process is increased,

$SR_{i,g} = SR_{i,g} + 1$. Using this success-based dynamic scheme, the number of individuals that an update process use is changed dynamically at each generation to reflect good performance without wasting extra computations. After determining this number, the two update processes are called as explained earlier in [Section 3.2](#).

4.2.2 Population Size Dynamic Reduction

After the first two stages of *d*DSF-EA are performed (differential evolution diffusion and success-based update processes), the last phase which is the dynamic population size reduction is performed. The goal is to use a dynamic scheme to change the population size, NP , dynamically for each generation based on logical search phases (exploration at the beginning and more exploitation at later stages of the search) and to improve the performance of the proposed algorithm. The general linear reduction scheme as proposed originally in [\[49\]](#) and L-SHADE algorithm [\[40\]](#) is used to reduce the population size as shown earlier in [Eq. \(1.27\)](#). Using this equation, the population is set with a large value at the beginning of the search which equals to $NP_{\max}$, i.e. $NP_{g_0} = NP_{\max}$. Then, the size at next generation, NP_{g+1} , is computed using [Eq. \(1.27\)](#) and the $(NP_g - NP_{g+1})$ worst individuals based on function values are eliminated from the population. Hence, the size is decreased to reach the minimum size denoted as $NP_{\min}$ which equals to the smallest possible value that is needed to perform the mutation strategy, which is 4 in this case, using “current-to-*p*best”.

4.2.3 Putting it all together

The full pseudocode of d DSF-EA is presented in [Pseudocode A. 3](#). The algorithm starts by the initialization procedure as given in Lines 1-7 in which a large population space is generated with size $NP_{\max}$. Next, new generation of offspring individuals is generated using the differential evolution diffusion process as presented earlier in subsection 3.3.1 and line 9. After that, the success-based update process is called which assigns an appropriate participation ratio for each update process as shown in Lines 10-19. At the end of the optimization loop, the population size reduction routine is performed to determine the reduced population size for the next generation as Lines 20-24 present.

4.3 Experimental Results

4.3.1 Parametric Setup and Algorithm Complexity

The d DSF-EA algorithm is tested on IEEE CEC'2014 benchmark problems [\[13\]](#). This benchmark set is explained earlier in subsection 3.4.1. The control parameter settings are set as follows:

- As d DSF-EA is an extension of SFS-DP_{DE-GW}, the same control parameter settings are used
- For population size dynamic reduction: $NP_{\max}$ and $NP_{\min}$ are set to $15 \times D$ and 4, respectively.

The algorithm was coded using Matlab 2014 and was run on an Intel® Core™ i7 CPU at 3.4 GHz speed with 8 GB RAM. The complexity of d DSF-EA is also

computed as presented in [Table C. 2](#) of Appendix C which shows $T_1, \hat{T}_2$ and $(\hat{T}_2 - T_1)/T_0$ for different dimensions following the guidelines of the used benchmark.

4.3.2 Compared Algorithms

In this section, the performance of d DSF-EA is compared with other state-of-the-art algorithms as well as other reported results from the literature as shown below:

1. SaDE: Self-adaptive differential evolution algorithm [\[42\]](#)
2. JADE: Adaptive differential evolution with optional external archive [\[41\]](#)
3. SHADE: Success-history based parameter adaptation for differential evolution [\[39\]](#)
4. L-SHADE: Differential evolution with linear population size reduction [\[40\]](#)
5. SFS: Stochastic fractal search [\[86\]](#)
6. SFS-DP_{DE-GW}: differential evolution with stochastic fractal search

For all the contestant algorithms, unless otherwise stated, we employ the best suited parametric setup for the employed benchmark suite, chosen with guidelines from their respective literature. The rest of the parameters are adaptively set during the search. All of the DE variant algorithms were started from the same initial population in each run to attribute any differences in the performance to internal search operators of the algorithms.

4.3.3 Results on Numerical Benchmark

The 30 benchmark problems are tested on $50D$. In order to evaluate how the performance of the proposed algorithm adapts with scaling the search space, a scalability test at $100D$ is also provided for all the functions. Following the guidelines provided in the IEEE CEC'2014 special session and competition [13], the maximum number of function evaluations (FEs) was set to 5×10^5 for $50D$ and 1×10^6 for $100D$.

A comparison of mean and the standard deviation of the best-of-run errors for 51 independent runs of all the contestant algorithms on the 30 benchmark problems for $D=50$ and $D=100$ is presented Table D. 3 and Table D. 4 respectively. A two-sided Wilcoxon rank sum test [87] is used to evaluate the statistical significance of the observed differences between the contestant algorithms. More details about this test can be found in Appendix D. According to the obtained Wilcoxon's test, the results are summarized as “ $w/t/l$ ” in Table 4. 1 and also the last row of Table D. 3 and Table D. 4, to imply that the contestant algorithm wins in w functions, ties in t functions, and loses in l functions, in a pairwise comparison with d DSF-EA algorithm.

As indicated in Table D. 3, and considering the mean of the error values for $50D$ problems and as noted from results of the Wilcoxon test, d DSF-EA shows a statistically significant performance when compared with all the contestant algorithms in most of the functions. The comparison of d DSF-EA indicates that the proposed algorithm was at least as good as the other competent algorithms in 21 functions on $50D$. L-SHADE, the announced winner of the IEEE CEC special

session and competition [55], performs better than d DSF-EA in 4 functions, shows a similar performance in 12 functions, and shows an inferior performance in 14 functions. A careful scrutiny of the results of the comparison with the proposed algorithm, SFS-DP_{DE-GW}, shows that d DSF-EA wins in 14 functions, shows equal performances in 10 functions and loses in 6 functions. Compared to the original components (SFS and SHADE algorithms), the results reveal that d DSF-EA advances the performance in an effective manner as the number of tie and inferior performances are decreased when SFS-DP_{DE-GW} is compared to these two algorithms. Also, the number of wins is stable or decreased as in a pairwise comparison with SFS. This demonstrates the effectiveness of the introduced powerful amalgam in d DSF-EA which advances the performance of the evolutionary search, and balances the exploration and exploitation phases with an appropriate use of function evaluations.

A scalability study using 100D was performed following the specifications of the IEEE CEC'2014 special session and competition as can be seen in Table D. 4. The relative performance of d DSF-EA was not substantially decreased with increasing the dimensionality to 100D. As revealed by Table 4. 1, d DSF-EA obtained the best performance over the majority of these functions when compared to the other contestant algorithms. It was able to obtain better results in 13 functions compared to the announced winner of the benchmark suits (L-SHADE), equal performance in 14 functions, and a lower performance in 4 functions. Inspecting the results of SFS-DP_{DE-GW} at 100D as shown in Table 4. 1 shows that d DSF-EA was able to obtain statistically significant results in 17 functions, equal performance in 10 functions and an inferior performance in 3

functions. It is clearly noticed that d DSF-EA advances performance on $100D$ compared to SFS-DP_{DE-GW}, and this reveals the power of the success-based update strategy to better balance the exploration and exploitation, during the search. Similar as $50D$, d DSF-EA shows a competitive performance in solving most of the hybrid and composition functions with clear difference in comparison to the contestant algorithms.

Table 4. 1: Comparison of d DSF-EA with state-of-the-art algorithms on $D = 50$ and 100 using the Wilcoxon rank-sum test (significantly, $p < 0.05$)

Vs. d DSF-EA		$D = 50$	$D = 100$
SaDE	+ (win)	0	2
	= (equal)	2	1
	- (lose)	28	27
JADE	+ (win)	2	4
	= (equal)	5	3
	- (lose)	23	23
SHADE	+ (win)	2	7
	= (equal)	6	5
	- (lose)	22	18
L-SHADE	+ (win)	4	4
	= (equal)	12	13
	- (lose)	14	13
SFS	+ (win)	0	2
	= (equal)	8	8
	- (lose)	22	20
SFS-DP _{DE-GW}	+ (win)	6	3
	= (equal)	10	10
	- (lose)	14	17

4.4 Conclusion

In this chapter, a decremental differential stochastic fractal evolutionary algorithm (*d*DSF-EA) is proposed which is a new improved extension of the proposed algorithm, SFS-DP_{DE-GW}. Investigations found that there is no balance between exploration and exploitation phases in SFS-DP_{DE-GW} due to inefficient use of function evaluations. This problem was curtailed by proposing a new success-based scheme to dynamically adjust the participation ratio of each stage in *d*DSF-EA with balancing the exploration/exploitations features of the search. The proposed work presents a three-stage optimization algorithm: differential evolution diffusion process, success-based update process and population size dynamic reduction. The same engine of SFS-DP_{DE-GW} is adopted in the first stage and is used along with the diffusion process to better generate new offspring. The second stage of the proposed work presents a dynamic success-based mechanism to call the update processes in this new amalgam. The proposed success-based scheme assigns an appropriate participation ration for each update process based on its previous success. Hence, utilizing the allowed function evaluations in an effective manner which reflects the quality of newly generated individuals. The performance of the proposed algorithm is tested on a challenging set of 30 benchmark problems taken from the IEEE CEC'2014 single objective optimization special session and competition. Simulation results affirm the fact that the new hybrid algorithm significantly advances the efficiency of the search over SFS-DP_{DE-GW} and L-SHADE algorithm (winner of competition), and other state-of-the-art algorithms.

Chapter 5 Summary of Part I

Despite the fact that the differential evolution algorithm has been shown as powerful techniques for solving real-valued optimization problems, it can reach stagnation scenarios and premature convergence easily when solving complex optimization problems. In this part of the thesis, a potential effort has been made to provide solutions for this problem using a synergistic style by developing new algorithms based on hybridization and combination of the strengths of differential evolution and stochastic fractal search. Two hybrid algorithms are introduced in this part namely, SFS-DP_{DE-GW} and *d*DSF-EA which are presented in Chapter 3 and 4 respectively. These two hybrid algorithms merge the search properties and capabilities to offer new algorithms with new features that are superior in terms of robustness and resiliency to their constituent algorithms.

In SFS-DP_{DE-GW}, we merge the DE operators in the diffusion process of SFS, hence the process is called diffusion differential evolution which uses SHADE mutation and a mixture of two Gaussian walks to guide the evolutionary search of the new amalgam. Despite the fact that this new algorithm enhances the performances of its two components: SFS and SHADE algorithms, it lacks a good balance between exploration and exploitation phases. Motivated by this observation and to solve this shortcoming, *d*DSF-EA algorithm is introduced, which is an enhanced version of SFS-DP_{DE-GW} with two more additions. These additions include: success-based update process and a dynamic population size reduction to improve the performance. Instead of using the whole population space for the update process as in SFS-DP_{DE-GW} which consumes extra computations, the participation ratio of each update process is changed

dynamically. This depends on the success rate of the previous generations, and hence reflects the appropriate use of function evaluations.

5.1 Comprehensive Comparison of the Methods in Part I

In this section, we present an overall performance comparison of the proposed methods in part I (SFS-DP_{DE-GW} and *d*DSF-EA) together with state-of-the-art DE algorithms as well as other best reported results from the literature on the used benchmark suite. The compared algorithms can be distinguished into two different classes of algorithms. The first seven algorithms (dynNP-jDE, OXDE, JADE, SaDE, SHADE, L-SHADE, and SFS) include pure homogeneous evolutionary algorithms with single and/or adaptive mutation scheme algorithms. The last five algorithms (EPSDE, CoDE, CoBiDE, UMOEAs, and AMALGAM) include hybrid algorithms and/or an ensemble of mutations schemes algorithms. For all the contestant algorithms, unless otherwise stated, the best suited parametric setup for the employed benchmark suite is employed, chosen with guidelines from their respective literature. The rest of the parameters are adaptively set during the search.

1. dynNP-jDE: population size reduction for the differential evolution algorithm [\[49\]](#)
2. OXDE: enhancing the search ability of differential evolution through orthogonal crossover [\[89\]](#)
3. SaDE: Self-adaptive differential evolution algorithm [\[42\]](#)
4. JADE: Adaptive differential evolution with optional external archive [\[41\]](#)

5. SHADE: Success-history based parameter adaptation for differential evolution [39]
6. L-SHADE: Differential evolution with linear population size reduction [40]
7. SFS: Stochastic fractal search [86]
8. EPSDE: Differential evolution algorithm with ensemble of parameters and mutation strategies [50]
9. CoDE: differential evolution with composite trial vector generation strategies and control parameters [51]
10. CoBiDE: Differential evolution based on covariance matrix learning and bimodal distribution parameter setting [90]
11. UMOEAs: united multi-operator evolutionary algorithms [85]
12. AMALGAM-SO: Self-Adaptive Multimethod Search [84]
13. SFS-DP_{DE-GW}
14. *d*DSF-EA

The results of the Wilcoxon rank sum test are summarized first, in which a pairwise comparison between *d*DSF-EA and each of the contestant algorithms is performed, as presented in Table 5. 1 for 50D and 100D. *d*DSF-EA shows a competitive performance compared to SFS-DP_{DE-GW} and hence it is chosen as the best performing algorithm for the Wilcoxon test. As indicated to Table 5. 1, *d*DSF-EA obtained the best performance over most of these functions when compared to the other algorithms. It was able to obtain better results in 14 and 13 functions compared to winner of CEC'2014 competition (L-SHADE) on 50D and 100D, respectively, while it shows inferior performance in 4 functions

Table 5. 1: Comparison of d DSF-EA with state-of-the-art algorithms on $D = 50$

and 100 using the Wilcoxon rank-sum test (significantly, $p < 0.05$)

Vs. d DSF-EA		$D = 50$	$D = 100$
dynNP-jDE	+ (win)	0	7
	= (equal)	7	4
	- (lose)	23	19
OXDE	+ (win)	0	2
	= (equal)	2	3
	- (lose)	28	25
SaDE	+ (win)	0	2
	= (equal)	2	1
	- (lose)	28	27
JADE	+ (win)	2	4
	= (equal)	5	3
	- (lose)	23	23
SHADE	+ (win)	2	7
	= (equal)	6	5
	- (lose)	22	18
L-SHADE	+ (win)	4	4
	= (equal)	12	13
	- (lose)	14	13
SFS	+ (win)	0	2
	= (equal)	8	8
	- (lose)	22	20
EPSDE	+ (win)	3	4
	= (equal)	2	1
	- (lose)	25	25
CoDE	+ (win)	3	4
	= (equal)	2	2
	- (lose)	25	24
CoBiDE	+ (win)	3	3
	= (equal)	6	4
	- (lose)	21	23
UMOEAs	+ (win)	2	10
	= (equal)	8	4
	- (lose)	20	16
AMALGAM-SO	+ (win)	3	6
	= (equal)	3	2
	- (lose)	24	22
SFS-DP _{DE-GW}	+ (win)	6	3
	= (equal)	10	10
	- (lose)	14	17

on both dimensions. Compared to well-known successful hybrid algorithms: UMOEAs and AMALGAM-SO, d DSF-EA shows competitive performance on both dimensions. UMOEAs and AMALGAM-SO fail in 20 and 24 functions out

of 30 compared to d DSF-EA algorithm, while they win in 2 and 3 functions receptively on $50D$. Similar to $50D$, d DSF-EA shows a superior performance with clear differences when solving $100D$ compared to the contestant algorithms. Inspecting the results on this table reveals that the proposed algorithms in this part show a competitive performance and d DSF-EA is better than its origin (SFS-DP_{DE-GW}) which wins in 14 and 17 functions on $50D$ and $100D$, respectively. This proofs the effectiveness and reliability of the proposed algorithm.

Furthermore, an extensive statistical analysis was added for the purpose of evaluating the statistical significance of the observed performance differences. For an input of n algorithms, this analysis employs a statistical test procedure to rank the performance of each compared algorithm. The test highlights whether there are statistically significant differences in the performance ranking of at least one pair of these compared algorithms. For such purpose, the aligned Friedman test [91] [92] is a non-parametric multiple comparison test that was used to test the differences between the above 14 compared algorithms (including d DSF-EA and SFS-DP_{DE-GW}), and judge the performance of the different samples. Table 5. 2 and Table 5. 3 presents the rankings achieved by aligned Friedman test for $50D$ and $100D$, respectively. The last two rows in Table 5. 2 and Table 5. 3 show the test-statistic for this test which is distributed according to chi-square with 13 degrees of freedom and the corresponding measured p -value, respectively. Friedman test assigns the lowest rank to the best performing algorithm. As can be seen in this table, the lowest rank score was obtained by d DSF-EA (rank@ $50D$ =131.0500, rank@ $100D$ =133.3833) with a clear difference compared to other algorithms. SFS-DP_{DE-GW} is ranked the second-best

performing algorithm for 50D with a rank equals to 159.3000, while L-SHADE is ranked at the third place with a rank equal to 165.2333. For 100D, L-SHADE is ranked the second-best performing algorithm with a rank equals to 159.85 and UMOEAs is at the third place with a 167.3167 rank. As has been noticed in [Table 5.3](#), the performance of SFS-DP_{DE-GW} is degraded for 100D and is ranked at the 8th place.

Table 5.2: Average ranking of competitor algorithms, achieved by the Aligned Friedman test at $D=50$. (Best entry is marked in bold)

Algorithm	Aligned Friedman Ranking	Order
SaDE	335.7000	14
EPSDE	267.7500	13
OXDE	253.7667	12
Amalgam	239.6833	11
JADE	236.3167	10
SFS	224.3833	9
CoBiDE	199.6833	8
CoDE	197.5333	7
dynNP-jDE	184.4667	6
UMOEAs	176.5833	5
SHADE	175.5500	4
L-SHADE	165.2333	3
SFS-DP _{DE-GW}	159.3000	2
dDSF-EA	131.0500	1
Statistic	26.1057730	
p -value	0.01645335	

Table 5.3: Average ranking of competitor algorithms, achieved by the Aligned Friedman test at $D=100$. (Best entry is marked in bold)

Algorithm	Aligned Friedman Ranking	Order
-----------	--------------------------	-------

OXDE	310.7833	14
SaDE	273.5167	13
EPSDE	260.0667	12
SFS	236.1667	11
JADE	220.6167	10
CoDE	218.4667	9
SFS-DP _{DE-GW}	216.6333	8
Amalgam	201.5000	7
CoBiDE	193.3333	6
SHADE	182.2833	5
dynNP-jDE	173.0833	4
UMOEAs	167.3167	3
L-SHADE	159.8500	2
dDSF-EA	133.3833	1
Statistic	26.3687508	
<i>p</i> -value	0.01516272	

PART II

Improved Differential Evolution Algorithms based on Ensemble Parameter Adaptation

Chapter 6 Ensemble sinusoidal differential evolution: The LSHADE-EpSin algorithm

6.1 Introduction

The differential evolution algorithm is a powerful tool that successfully handled many optimization problems and applications [1][2]. Although its effectiveness and competitive performance has been demonstrated, many experimental studies and theoretical analysis [5][6] show that it is sensitive and highly dependent on the settings of its control parameters. These settings include scaling factor (F), crossover rate (CR), mutation/crossover strategy and population size (NP). The proposed mechanisms for control parameter calculation might have control parameters as well. Also, the rate of changing these control parameters might influence the performance. Motivated by this consideration, many studies have been proposed in the literature. Such studies introduce adaptive and self-adaptive methods to control the parameter settings of DE as have been explained earlier in subsection 2.3. Although these methods succeed in enhancing the performance of DE when solving a diverse set of optimization problems, locating the optimal solution is still a challenging task in most of these methods for complex optimization problems. Hence, there is still a room to explore new self-adaptive methods which reflects competitive performance compared to other existing approaches.

The adaptive and self-adaptive mechanisms can improve the DE performance if designed carefully by updating the parameter settings to suite the required

performance with different fitness landscapes. They have showed more reliable performance and faster convergence compared to the deterministic methods or trial-and-error methods without parameter control. In addition, these two types are applicable for different optimization problems and real-world applications because they can find the suitable settings of a specific problem based on the feedback of the evolutionary search. Motivated by these considerations, a novel self-adaptive method based on an efficient ensemble sinusoidal scheme to control the settings of the scaling factor was introduced. This ensemble is based on a mixture of sinusoidal formulas and a Cauchy distribution. Two sinusoidal formulas were proposed: non-adaptive sinusoidal decreasing adjustment and an adaptive history-based sinusoidal increasing adjustment. The objective behind using this new scheme is to have a good balance between exploration and exploitation processes during the search benefiting from the periodicity of the sinusoidal function.

In the context of developing an adaptive sinusoidal approach to adapt the control parameter settings in DE, the proposed approach falls in the first class which uses an adaptive scheme to control the settings of the *freq* parameter in the sinusoidal formula. The prior successful knowledge is used to adjust the *freq* parameter. This approach helps in fitting the sinusoidal curve to form more suitable settings of the scaling parameter. This new adaptation is embedded in L-SHADE, and the new proposed algorithm is namely LSHADE-EpSin. To enhance the evolutionary search of the proposed algorithm, a local search is used at later generations based on Gaussian walks. These new additions provide more suitable

settings compared to the L-SHADE algorithm's and provides a good balance between exploitation and exploration through successful information sharing.

The reminder of this chapter is organized as follows. The proposed LSHADE-EpSin is presented in Section 6.2 with detailed description. The experimental setup and simulation results are presented in Section 6.3. Finally, Section 6.4 summarizes the conclusions of this chapter.

6.2 LSHADE-EpSin: How it Works

In the proposed algorithm, LSHADE-EpSin, parameter adaptation of L-SHADE algorithm is further enhanced by using sinusoidal waves to adapt the scaling factor. In this adaptation scheme, an ensemble pool that consists of two sinusoidal waves and Cauchy distribution was used to adapt the scaling factor in each generation. Furthermore, a local search based on a Gaussian walk is used at later generations to enrich the evolutionary search. The following sub-sections explain the proposed algorithm in detail.

6.2.1 Ensemble of Parameter Adaptation

The parameter adaptation scheme adapts the scaling factor in which a Cauchy distribution and two sinusoidal formulas are used. The motivation behind using this mixture is to enrich the search by generating different values for the parameters in DE to promote the search toward the promising regions. One of the most important key features of using the sinusoidal formula to adapt the scaling factor is the ability to change the search direction. This change constitutes permitting incremental and decremental adjustments to the parameter values in a controlled fashion.

The use of sinusoidal adaptation is activated in the first half of generations which consists of two formulas: sinusoidal decreasing adjustment and sinusoidal increasing adjustment. This configuration is presented in Figure 6. 1. The reason behind using this mixture is that the sinusoidal formulas make it easy to periodically change the value of the scaling factor using the increasing and decreasing adjustments. Hence, the search direction can be easily drifted, and the exploration can be enhanced.

At each generation, $g \in \left[1, \frac{g_{\max}}{2}\right]$ where $g_{\max}$ is the maximum generation number, is adapted using a sinusoidal decreasing adjustment, as expressed in Eq. (2.8), or a sinusoidal increasing adjustment, as expressed in Eq. (2.9). The selection between these two adjustments is done randomly.

$$F_{i,g} = 0.5 \times \left(\sin(2\pi \times freq \times g + \pi) \times \frac{g_{\max} - g}{g_{\max}} + 1 \right) \quad (2.8)$$

$$F_{i,g} = 0.5 \times \left(\sin(2\pi \times freq_{i,g} \times g) \times \frac{g}{g_{\max}} + 1 \right) \quad (2.9)$$

where the $freq$ parameter in Eq. (2.8) is set to a fixed value while $freq_{i,g}$ in Eq. (2.9) is auto-adjusted using an adaptive scheme as shown in the following equation:

$$freq_{i,g} = Cauchy(\mu freq_{i,g}, 0.1) \quad (2.10)$$

The Cauchy distribution is used with a mean of $\mu_{freq_{i,g}}$ and a standard deviation equals to 0.1. At each generation g , the successful mean frequencies are stored in S_{freq} is updated at the end of each generation using the usual arithmetic mean.

Figure 6. 1: Sinusoidal adjustment configuration

For the second half of generations, $g \in \left(\frac{g_{\max}}{2}, g_{\max}\right]$, $F_{i,g}$ is adapted as originally suggested by L-SHADE using a Cauchy distribution (Eq. (1.14)). The key reason behind activating the Cauchy distribution to adapt $F_{i,g}$ in the second half of generations is in its ability to generate values that are best for invoking an exploitation phase during the search process. This helps to avoid obtaining values from the traditional exploitation in DE that often occurs when using a greedy mutation such as “current-to- p best” especially at later generations [41].

The crossover rate, $CR_{i,g}$, is adapted using a normal distribution for all the generations as proposed in L-SHADE. The successful values of $CR_{i,g}$ and $F_{i,g}$ which are able to generate trial vectors better than the parent vectors are recorded in S_F and S_{CR} , respectively. A memory is used to store the successful mean values for μF and μCR . At the end of each generation, the memory is updated where the index k determines the position to be updated and is incremented when new values are accepted into the memory. The two means μF and μCR at index k are modified using the weighted Lehmer mean as explained in Eq. (1.20) and Eq. (1.22). As the Cauchy adaptation is activated at the second half of generations for $F_{i,g}$, the aforementioned process is performed at this time only, while sinusoidal is used for the first half only.

6.2.2 Local Search

A local search based on Gaussian walks is used to enrich the evolutionary search of LSHADE-EpSin at later generations. As the population size reduction is used, this search is called when NP is decreased to reach 20. When this condition is satisfied during the run, the local search is activated. Firstly, 10 random individuals are initialized using Eq. (1.3) denoted as $X_i, i \in [1, 10]$. Next, a new individual, $X_{i,new}$, is generated using a Gaussian walk as presented earlier in SFS (subsection 3.2, Eq. (2.1)). The key reason behind using this Gaussian walk as a local search after extensive experiments is in its ability to generate new good individuals benefiting from the information of the obtained best individual. Also, it shows good performance to solve hybrid and composition functions. In Eq.

(2.1), the mean, μ_{BP} , for $\text{Gaussian}(\mu_{BP}, \sigma)$ is set to X_{best} while standard deviation, σ , is computed as shown in Eq. (2.3).

This local search is performed for a number of generations, denoted as G_{LS} . Then, the 10 worst individuals from the original population are selected as candidates for replacement. In case the newly generated individuals from local search are better than the selected individuals, the replacement occurs.

6.2.3 Putting it all together

The overall LSHADE-EpSin algorithm is presented in Pseudocode A. 4 of Appendix A. Lines 1-9 represent the initialization of the population and setup of the initial values for parameter settings. Then an optimization loop is started to generate a new offspring using “current-to- p best” mutation strategy and the introduced adaptation settings. Lines 11-22 describe the sinusoidal settings to adapt $F_{i,g}$ for the first half of generations, and then using Cauchy for the second half. Line 23 describes the adaptation of $CR_{i,g}$ using normal distribution. The “current-to- p best mutation” with binomial crossover is invoked in lines 24-37 and the successful $F_{i,g}$, $CR_{i,g}$ and $freq_{i,g}$ are stored in S_F , S_{CR} and S_{freq} , respectively. Next, μF , μCR and $\mu freq$ are updated in Lines 38-39. Next, the linear population size reduction is used as presented in Lines 40-43. When NP is decreased to 20, the local search method is called as shown in Lines 44-51.

6.3 Experimental Results

6.3.1 Numerical Benchmark

The performance of LSHADE-EpSin is evaluated using a set of problems presented in the CEC2016 competition on real-parameter single objective optimization. In this competition, the benchmark functions were reused from the CEC'2014 benchmark set [13]. More details on the benchmark suite can be found in subsection 3.4.1, and Table B. 1 in Appendix B.

6.3.2 Parametric Setup and Algorithm Complexity

The parameter values of LSHADE-EpSin algorithm are set as follows:

- The Memory size H is set to 5 with μF and μCR are both set to 0.5
- For sinusoidal adjustment: $freq$ is set and fixed to 0.5 for decreasing adjustment, and initial value of $\mu freq_{i,g_0}$ is set to 0.5 for increasing adjustment.
- For population size dynamic reduction: $NP_{\max}$ and $NP_{\min}$ are set to $18 \times D$ and 4, respectively.
- After extensive experiments, G_{LS} is set to 250 generations as it was an enough exploitation window for the entire search process to advance the evolutionary search. Less than this is not enough to utilize the used individuals to find better results, and it was observed that using more than this number will not add to the maximum utilization that can be obtained using this organization.

The algorithm was coded using Matlab 2014a and was run on a PC with an Intel CPU (3.40GHz) and 8GB RAM and windows 7. Algorithm complexity of LSHADE-EpSin is computed following the guidelines of the benchmark suite as explained in Appendix C. [Table C. 3](#) summarizes the complexity for the different dimensions.

6.3.3 Compared Algorithms

The performance of the L-SHADE-EpSin is compared with the performances of other state-of-the-art differential evolution algorithms listed below. The same control parameter values that were suggested in the original papers were used to run the aforementioned algorithms.

1. L-SHADE: differential evolution with linear population size reduction [\[40\]](#)
2. SHADE: differential evolution with success-history based parameter adaptation [\[39\]](#)
3. JADE: adaptive differential evolution with optional external archive [\[41\]](#)
4. EPSDE: differential evolution with an ensemble of parameters and mutation strategies [\[50\]](#)
5. DynNP-jDE: differential evolution with reduced population size [\[49\]](#)
6. CoDE: differential evolution with composite trial vector generation strategies and control parameters [\[51\]](#)

6.3.4 Experiments and numerical results

Each algorithm was run for $10,000 \times D$ functions evaluations for each of 51 independent runs. When the difference between best solution found and the optimal became less than 1×10^{-8} , the error was treated as 0. Table D. 5 presents the mean and standard deviation of the best-of-run errors for the 51 independent runs of all used algorithms on 30 benchmark problems on 50D. The best error value is marked in bold. This table shows that the LSHADE-EpSin obtained a performance that is at least as good as the other algorithms in 25 out of the 30 functions. LSHADE-EpSin performed better than the winner of the 2014 competition (L-SHADE), which has the best results in the literature for the CEC'2014 in most of the functions. This affirms the power of LSHADE-EpSin algorithm and the effectiveness of using the proposed sinusoidal ensemble pool of control settings and local search.

Moreover, the Wilcoxon's rank-sum test was used to judge the significance of the results [87]. Table 6. 1 summarizes the results of this test at a 0.05 significance level comparing each algorithm versus LSHADE-EpSin on the 30 functions for $D=30, 50$ and 100 . For each of the competitive functions in this table, “–” identifies the number of losses to LSHADE-EpSin, “+” marks the instances where the contestant algorithm exhibits a superior performance compared to LSHADE-EpSin, and “=” indicates that the performance difference between the competitor and LSHADE-EpSin is not statistically significant. As shown in this table, LSHADE-EpSin clearly has the best overall performance among all the variants and significantly improves upon the performance of L-SHADE algorithm as the number of dimension increases. It also demonstrates the

efficiency of using the ensemble pool of sinusoidal adjustments and Cauchy distribution to adapt the scaling factor which provides a balance between exploration and exploitation. Also, the use of local search based on Gaussian walk advances the performance to solve hybrid and composition functions.

Table 6. 1: Comparison of LSHADE-EpSin with state-of-the-art DE algorithms on $D = 30, 50, 100$ using the Wilcoxon rank-sum test (significantly, $p < 0.05$)

Vs. LSHADE-EpSin		$D = 30$	$D = 50$	$D = 100$
CoDE	+ (win)	1	2	1
	= (equal)	10	5	4
	- (lose)	19	23	25
dynNP-jDE	+ (win)	0	1	3
	= (equal)	10	8	5
	- (lose)	20	21	22
SaDE	+ (win)	0	0	1
	= (equal)	3	2	3
	- (lose)	27	25	26
JADE	+ (win)	0	2	3
	= (equal)	12	6	5
	- (lose)	18	22	22
SHADE	+ (win)	0	2	4
	= (equal)	11	10	6
	- (lose)	19	18	20
L-SHADE	+ (win)	3	2	2
	= (equal)	15	12	10
	- (lose)	12	16	18

6.4 Conclusion

In this chapter, we propose LSHADE-EpSin which is an improved version of the L-SHADE algorithm using two new additions. The algorithm introduces a novel method to adapt the scaling factor and a local search based on Gaussian walks. A

new ensemble pool of sinusoidal adjustments and Cauchy distribution is used to adapt the values of the scaling factor in each generation. This ensemble pool consists of a mixture of two sinusoidal formulas which are activated for the first half of generations, while Cauchy distribution is used for the second half. The key advantage is to promote exploration benefiting from the periodicity of the sine function, and then the exploitation using Cauchy distribution. For later generations, a Gaussian-walk-based local search is used to advance the performance to generate new better individuals. The proposed algorithm is tested on the IEEE CEC'2014 benchmark suite on real-parameter single objective optimization. When compared with other state-of-the-art DE algorithms including the best performing algorithm, it shows a very competitive performance.

Chapter 7 Ensemble sinusoidal differential evolution algorithm with niching reduction: EsDEr-NR algorithm

7.1 Introduction

As has been explained earlier, many theoretical analyses and experimental studies proved that the DE performance depends on the settings of its control parameters. These parameters are: scaling factor (F), crossover rate (CR), and population size (NP). There are few studies which investigate the population size settings in DE and EAs in general compared to other comprehensive studies that were developed for adapting the scaling factor and the crossover rate. The early studies for population resizing suggest the use of meta-level control parameters to adjust the population size in GA [93][94]. This method is practically difficult to use and time consuming due to the need of fine tuning the hidden meta-level parameters. Later in recent years, the deterministic rules are used to overcome the shortcomings of these previous methods. These rules are simple and do not require extra computation cost, and many EAs have been used them such as: GL-25 [95], IPSO [96], and SVPS (simple variable population size reduction) [97]. Recently, these methods are also used in DE and have been introduced to enhance the performance of DE for many recently developed algorithms such as dynNP-jDE [49], L-SHADE and all of its variants [40].

Despite the fact that these deterministic rules have been shown good performance without extra computations, they are resizing the population without considering

its effect to the evolutionary search. The common issue that most of the existing methods for resizing the population have in common is that decreasing the population size increases the possibility of stagnation and premature convergence [98][49]. Motivated by these observations, a novel approach to adjust the population size by using a niching-based reduction scheme is introduced. In this new mechanism, two separate niches are used before performing the population size reduction. The goal is to separate the population into two different niches with the use of Euclidean distance metric, and hence mark the elimination region in a proper manner. In the context of developing a niching-based population size reduction, the proposed approach falls in the first class. This new scheme is embedded in the previously introduced algorithm, LSHADE-EpSin, which has been explained in Chapter 6. The proposed algorithm utilizes an ensemble sinusoidal differential evolution with niching reduction, namely EsDE_r-NR.

The rest of this chapter is organized as follows. In Section 7.2, the new algorithm, EsDE_r-NR, is elaborated with detailed discussions. Simulation results are presented in Section 7.3 with comparative comparison. Finally, concluding remarks are summarized in Section 7.4.

7.2 EsDE_r-NR: How it Works

EsDE_r-NR algorithm possesses the following three main procedures:

- Evolutionary search with ensemble sinusoidal parameter adaptation as introduced in LSHADE-EpSin.
- Restart strategy or local search, as introduced in LSHADE-EpSin and
- Niching-based population size reduction.

Each of the aforementioned processes provides its strength to the others to complement their work toward boosting the evolutionary search. In the context of developing a niching-based population size reduction, the proposed approach is the first work which uses two separate niches for the purpose of performing population size reduction. The first two processes are explained earlier in the Chapter 6 (Section 6.2). The following sub-sections explain the proposed reduction approach in detail with overall algorithm.

7.2.1 Niching-based Population Size Reduction

In L-SHADE [40] and our proposed L-SHADE-EpSin algorithm, the population reduction starts at the first generation and continues till the end. This mechanism has two disadvantages. Firstly, starting the reduction at the first generation does not give the required time for the individuals to evolve well. This means wasting many function evaluations to initialize the individuals and then delete them immediately without benefiting from their information yet. At first generations, all the individuals are considered starter individuals and the meaning of reduction is not satisfied yet. Adding to that, this will be wasting function evaluations without any benefit. The second thing is that the current reduction methods including linear population size reduction increases the possibility of premature convergence while decreasing the population size. This is due to the fact that the existing methods eliminate the worst individuals based on their function values. Hence the diversity is lost, and the population converges to best individuals especially at later generation.

To overcome the aforementioned shortcomings, a novel method namely niching-based population size reduction is introduced. Reduction in this method starts after the first half of the function evaluations has been consumed (or after the first half of generations). This will give all the individuals the required time to evolve and hence an equal opportunity to evolve before starting the reduction. As a result, it guarantees the proper use of function evaluations without wastage. Next, two separate niches are formed to select the elimination region effectively before performing the reduction.

Firstly, the best individual, denoted as $X_{best,g}$, is marked. Then, the Euclidian distance, denoted as $ED_i(X_{i,g})$, is computed for every other individual in the current population as follows:

$$ED_i = ED_i(X_{i,g}, X_{best,g}) = \sqrt{\sum_{j=1}^D (x_{i,g}^j - x_{best,g}^j)^2} \quad (2.11)$$

After that, the individuals are sorted in an ascending order according to their Euclidean distances and the two niches are formed with a size equal to $\frac{NP}{2}$.

Figure 7. 1 illustrates this framework. The first niche is filled with the best individual, $X_{best,g}$, and its smallest Euclidean distance neighbours while the second niche is filled by the remaining individuals. Since the population size is started to decrease after the first half of generations, the two niches sizes are also decreased. The individuals to be eliminated are selected from the second niche while the first niche is preserved. The key reason behind that is to preserve the best individual and its neighbourhood regions from elimination to keep the

evolutionary search progressing, while separate the other individuals which are good candidate for elimination.

Figure 7. 1: An illustration for niching-based population reduction framework

Next, the simple deterministic rule, as proposed to L-SHADE and LSHADE-EpSin, which is presented in Eq. (1.27), is used to compute the reduced population size at the next generation. The number of individuals to be eliminated is then determined which is equal to $NP_g - NP_{g+1}$ where $g = \frac{g_{\max}}{2} + 1$. Those individuals are selected based on a different criterion that satisfies a proper elimination. In this criterion, the best individual is selected in the second niche to calculate the Euclidean distance for the other remaining individuals in this niche. Then, the individuals are sorted in an ascending order and the ones which have the smallest Euclidean distance are marked as an elimination region as in Figure 7. 1. Hence, the neighbourhood regions of the best individual, i.e. the individuals which are very similar to the best individual in the second niche are eliminated.

Therefore, this alleviates the loss of diversity of population while reducing the population in an effective manner.

7.2.2 Putting it all together

EsDE_r-NR is an enhanced version of LSHADE-EpSin with the addition of niching-based population size reduction. A full pseudocode is presented in [Pseudocode A. 5](#). Lines 1-9 represent the initialization of the population and setup of the initial values for the parameter settings. In Line 11, the algorithm starts by the evolutionary search as presented in LSHADE-EpSin using “current-to-*p*best” mutation strategy with the proposed ensemble sinusoidal parameter settings. Next, the niching-based population size reduction is started to reduce the population that started from the second half of generation as presented in Lines 13-25 with the use of two niches. When NP is decreased to 20, the local search based on Gaussian walk is called as shown in Lines 26-27.

7.3 Experimental Results

The EsDE_r-NR algorithm is also tested using IEEE CEC’2014 benchmark problems as has been explained in subsection 6.3.1 and [Table B. 1](#) in Appendix B. The following sub-sections present a comparative study with its statistical results, comparing the results of a EsDE_r-NR with other state-of-the-art algorithms.

7.3.1 Parametric Setup and Algorithm Complexity

There are no new parameters in EsDE_r-NR, hence the same parameter settings as has been used in LSHADE-EpSin (subsection 6.3.2) are employed. The algorithm was coded using Matlab 2014a and was run on a PC with an Intel CPU (3.40GHz) and 8GB RAM and Windows 7. The computational complexity of EsDE_r-NR algorithm is calculated as described in Appendix C for all the dimensions (10D, 30D, 50D and 100D). The algorithm complexity is presented in Table C. 1 which is reflected by $\hat{T}_2, T_1$ and $(\hat{T}_2, -T_1)/T_0$. Referring to Table C. 1 and Table C. 4, the algorithm complexity for EsDE_r-NR is comparable to LSHADE-EpSin without clear differences. A slight difference in such complexity is due to the computation of the Euclidean distances for the proposed dynamic reduction. The difference is slight only due to the efficient way of doing these computations by getting rid of the for-loops and the use of vectorization in our code to save computation time.

7.3.2 Compared Algorithms

A comprehensive comparison is made to compare the performance of EsDE_r-NR with the best performing algorithms in literature. These comparisons are split into two groups. In the first group, the performance of EsDE_r-NR was compared with state-of-the-art DE algorithms as follows. Those compared algorithms will be referred to as the first group in the results tables. It is important to compare the results with LSHADE-EpSin as the same engine was used to introduce EsDE_r-NR and with the recent DE algorithms. In SinDE algorithm, the “DE/rand/1” is used with a sinusoidal increasing adjustment to adapt both scaling factor and

crossover rate. Fixed values are used for their parameter settings for frequency and population size is set to 40.

1. CoDE: differential evolution with composite trial vector generation strategies and control parameters [51]
2. SHADE: success-history based parameter adaptation for differential evolution [39]
3. L-SHADE: improving the search performance of SHADE using linear population size reduction [40]
4. SinDE: differential evolution with sinusoidal adjustment for scaling factor and crossover rate [99]
5. LSHADE-EpSin: an ensemble sinusoidal parameter adaptation incorporated with L-SHADE

The control parameter values for these algorithms were used as suggested in the respective publications.

The following algorithms were used for the second group of the comparisons. These algorithms include state-of-the-art EAs and well-known recently developed amalgams such as CMA-ES variants which use population resizing method and show very superior performance to solve other benchmark suites. A new version of the famous mean-variance mapping optimization, MVMO algorithm is added in the comparisons. Moreover, UMOEAsII is also added as this algorithm obtained a competitive performance as has been introduced recently.

6. GaPADE: Gaussian adaptation-based parameter adaptation for differential evolution [100]
7. MVMO: Evaluating the mean-variance mapping optimization [101]
8. UMOEAsII: united multi-operator evolutionary algorithms-II [102]
9. IPOP-aCMA-ES: an advanced evolution strategy with covariance matrix adaptation integrated with an occasional restart strategy and increasing population size [103]
10. iCMAES-ILS: hybrid CMA-ES method which combines IPOP-CMA-ES and an iterated local search method [104]

For GaPADE algorithm, the population size is set to 100. For MVMO, the archive size is set to 25 and the population size is set to 100 for 50D and 150 for 100D. For UMOEAsII, the tournament selection is randomly set to 2 or 3 with $PS_1=PS_2=100$ and $CS=100$. For IPOP-aCMA-ES, the initial population size is set to $\lambda_0 = 4 + \lfloor a \ln(D) \rfloor$ and the maximum population size is set to 200. For iCMAES-ILS, the initial step size is set to 120.

7.3.3 Results on Numerical Benchmark

The mean and standard deviation of the best-of-run errors for 51 independent runs of the first group of algorithms on the 30 benchmark problems for 50D, and 100D are given in Table 6 and Table 7. The Wilcoxon's rank-sum test for independent samples [87], is used in order to judge the significance of the results. More details about this test can be found in Appendix D. According to the obtained Wilcoxon's test, the results are summarized as “w/t/l” in the last row of each of Table D. 6 and Table D. 7, to imply that the contestant algorithm wins in

w functions, ties in t functions, and loses in l functions, in a pairwise comparison with each of the other algorithms. A summary of these results is given in [Table 7. 1](#) for the both groups. As noted from the mean of the error values for 50D problems in [Table D. 6](#) and [Table D. 7](#) and considering the results of the Wilcoxon test, EsDER-NR outperforms all the other DE contestant algorithms in a statistical significant manner. The comparison of EsDER-NR against the first category of algorithms on 50D indicates that the proposed algorithm was at least as good as the other competent algorithms in 21 functions. The second performing algorithm was LSHADE-EpSin where it performs better than EsDER-NR in 3 functions, shows a similar performance in 20 functions, and shows an inferior performance in 7 functions. L-SHADE is ranked at the third place after L-SHADE-EpSin in which it shows a similar performance in 13 functions, wins in just one function and loses in 16 functions. EsDER-NR showed competitive performance compared to SinDE algorithm where the later algorithm fails in 25 functions and has equal performances in 5 functions. This reveals the effectiveness of using the proposed sinusoidal parameter adaptation with the use of ensemble pool.

A careful scrutiny of the results of the comparison with the state-of-the-art DE algorithms on 100D as presented in [Table D. 7](#) shows that the performance of EsDER-NR is not substantially decreased with increasing the dimensionality. The use of the proposed niching-based population size reduction shows better performance when increasing the dimensions as presented in [Table 7. 1](#) with comparison of LSHADE-

EpSin. The latter shows an inferior performance in 12 functions while it never showed a better performance at 100D when compared to EsDER-NR.

The comparison of EsDER-NR against the second category of algorithms on 50D as presented in [Table D. 8](#). The results of these tables show that the proposed algorithm is competitive compared to the other algorithms including CMA-ES variants and best performing, UMOEAsII. EsDER-NR obtained superior performance in 18 functions compared to iCMAES-ILS algorithm. iCMAES-ILS performs better than EsDER-NR in 5 functions, shows a similar performance in 6 functions, and shows an inferior performance in 17 functions. The second performing algorithm is UMOEsII in which it showed similar performance in 14 functions, wins in 2 functions and loses in 14 functions. Considering the 100D, EsDER-NR algorithm also shows competitive performance as presented in [Table D. 9](#). EsDER-NR performs better than iCMAES-ILS in 15 functions and obtains an equal performance in 7 functions. When compared to UMOEAsII, the latter wins in 4 functions, ties in 11 and loses in 15. The result tables reveal that the proposed algorithm is able to show competitive performance when compared to other best performing algorithms in the literature.

Table 7. 1: Comparison of EsDEr-NR with state-of-the-art algorithms on $D = 50, 100$ using the Wilcoxon rank-sum test (significantly, $p < 0.05$)

Vs. EsDEr-NR		$D = 50$	$D = 100$
CoDE	+ (win)	3	2
	= (equal)	2	4
	- (lose)	25	23
SHADE	+ (win)	1	4
	= (equal)	9	7
	- (lose)	20	19
L-SHADE	+ (win)	1	1
	= (equal)	13	13
	- (lose)	16	16
SinDE	+ (win)	0	1
	= (equal)	5	7
	- (lose)	25	22
LSHADE-EpSin	+ (win)	3	0
	= (equal)	20	18
	- (lose)	7	12
GaPADE	+ (win)	0	1
	= (equal)	11	7
	- (lose)	19	22
MVMO	+ (win)	2	4
	= (equal)	2	4
	- (lose)	26	28
UMOEAsII	+ (win)	2	4
	= (equal)	14	15
	- (lose)	14	11
IPOP-aCMA-ES	+ (win)	6	8
	= (equal)	7	8
	- (lose)	17	14
iCMAES-ILS	+ (win)	5	7
	= (equal)	6	7
	- (lose)	17	16

7.4 Conclusion

This chapter presents a new enhanced algorithm based on the previously discussed algorithm, LSHADE-EpSin. The new algorithm, named ensemble sinusoidal differential evolution with niching reduction, EsDEr-NR, consists of three main operations: evolutionary search with the use of ensemble sinusoidal adaptation, local search based on Gaussian walk and a new population size reduction. The main goal is to enhance the currently used reduction methods in the literature. The novel method, namely niching-based population size reduction, uses two niches to determine the best region for elimination. Using the niche-based reduction scheme, it will be highly likely that the population size is reduced in an effective manner. The IEEE CEC'2014 is used to validate the performance of EsDEr-NR algorithm. When compared with other state-of-the-art algorithm from the literature including competitive CMA-ES variants, it shows a very competitive performance and advances the search when increasing the dimensions.

Chapter 8 Summary of Part II

Differential evolution is highly dependent on the control parameter settings. If the parameters have been designed carefully, the performance of DE can be improved. Despite the fact that there are many adaptive methods that have been introduced in the literature to control the settings of the scaling factor and crossover rate parameters, there is still a room to investigate new techniques for enhancement. In the second part of the thesis, this topic will be targeted and new improved differential evolution algorithms that are based on novel parameter adaptation will be introduced. The goal is to propose new efficient methods to adapt the control settings of DE compared to existing approaches.

In Chapter 6, a novel ensemble sinusoidal parameter adaptation for the settings of scaling factor was introduced. This algorithm, LSHADE-EpSin, uses the same engine of L-SHADE algorithm with a new adaptation technique and a local search method to enrich the evolutionary search at later generations. This ensemble is based on a mixture of sinusoidal formulas and a Cauchy distribution. Two sinusoidal formulas were proposed: non-adaptive sinusoidal decreasing adjustment and adaptive sinusoidal increasing adjustment. The objective behind using this new scheme is to have a good balance between exploration and exploitation processes during the search. For the first half of generations, the sinusoidal scheme is activated which promote the exploration based on the periodicity of the sinusoidal function. Next, the Cauchy distribution is used which helps in exploitation phase due to its stability. The proposed algorithm falls in the first class which proposes the use of sinusoidal function and Cauchy distribution in an ensemble pool to adapt the settings of scaling factor.

Experimental results show that the proposed algorithm achieved superior performance compared to other best performing algorithms in the literature.

In the recent past, there is relatively fewer studies which control the settings of population size in differential evolution, compared to huge works that have been undertaken for the settings of the scaling factor and crossover rate. Most of the existing approaches for resizing or controlling the population size suffers from the possibility of increasing the stagnation and premature convergence. This is due to the fact that all of these methods select the worst individuals as cardinalate for elimination without paying attention to other considerations. In Chapter 7, these issues were addressed, and a novel niching-based population size reduction was introduced. That algorithm, $EsDE_r$ -NR, is a continuous enhanced version of the LSHADE-EpSin algorithm. Using this method, it will be highly likely that the population size will be reduced in an effective manner using two separate niches with the Euclidean distance metric. The results show that the new method performs favourably and more efficient compared to other state-of-the-art algorithms especially when dealing with scaled-up dimensions.

8.1 Overall Comparison of the Methods in Part I-II

In this section, an overall comparison of the newly introduced algorithms in part I and II are introduced, especially that the same benchmark suite (IEEE CEC'2014) is used to validate their performances. It is better to present a comprehensive comparison with other state-of-the-art algorithms including best performing algorithms in the literature. In part I of this thesis, two new improved hybrid DE algorithms were introduced. These are: $SFS-DP_{DE-GW}$ and $dDSF-EA$.

In part II of the thesis two enhanced DE algorithms based on new adaptive control parameter settings were introduced, which are: LSHADE-EpSin and EsDE_r-NR. These 4 algorithms are compared with the following 18 algorithms including state-of-the-art adaptive DE algorithms, constituting component algorithms that have been used to develop the proposed algorithms, successful hybrid amalgam (AMALGAM-SO and UMOEAs), winner of the IEEE CEC'2014 competition (L-SHADE), powerful CMA-ES variants and the newly best performing algorithms on the same benchmark (UMOEAs and UMOEAsII).

1. SaDE: Self-adaptive differential evolution algorithm [\[42\]](#)
2. JADE: Adaptive differential evolution with optional external archive [\[41\]](#)
3. SHADE: Success-history based parameter adaptation for differential evolution [\[39\]](#)
4. L-SHADE: Differential evolution with linear population size reduction [\[40\]](#)
5. SFS: Stochastic fractal search [\[86\]](#)
6. EPSDE: Differential evolution algorithm with ensemble of parameters and mutation strategies [\[50\]](#)
7. CoDE: Differential evolution with composite trial vector generation strategies and control parameters [\[51\]](#)
8. dynNP-jDE: population size reduction for the differential evolution algorithm [\[49\]](#)
9. OXDE: enhancing the search ability of differential evolution through orthogonal crossover [\[89\]](#)

10. CoBiDE: Differential evolution based on covariance matrix learning and bimodal distribution parameter setting [90]
11. UMOEAs: united multi-operator evolutionary algorithms [85]
12. AMALGAM-SO: Self-Adaptive Multimethod Search [84]
13. SinDE: differential evolution with sinusoidal adjustment for scaling factor and crossover rate [99]
14. GaPADE: Gaussian adaptation-based parameter adaptation for differential evolution [100]
15. MVMO: Evaluating the mean-variance mapping optimization [101]
16. UMOEAsII: united multi-operator evolutionary algorithms-II [102]
17. IPOP-aCMA-ES: an advanced evolution strategy with covariance matrix adaptation integrated with an occasional restart strategy and increasing population size [103]
18. iCMAES-ILS: hybrid CMA-ES method which combines IPOP-CMA-ES and an iterated local search method [104]
19. SFS-DP_{DE-GW}
20. *d*DSF-EA
21. LSHADE-EpSin
22. EsDE_r-NR

As used in Chapter 5, the aligned Friedman test is used to judge the performance of the above 22 algorithms. Table 8. 1 and Table 8. 2 present average ranking of contestant algorithms achieved by the aligned Friedman test for 50D and 100D, respectively. The last two rows show the test-statistic for this test which is distributed according to chi-square with 21 degrees of freedom and the

corresponding measured p -value, respectively. As can be noted from these tables, the best rank is for our proposed algorithm EsDEr-NR for both 50D and 100D. UMOEAsII algorithm is ranked at 2nd and 3rd places after our proposed algorithms.

These tables show that the rank for our proposed adaptive algorithms in part II EsDEr-NR, LSHADE-EpSin) are better than the two hybrid algorithms in part I (SFS-DP_{DE-GW}, dDSF-EA). For 50D, the proposed algorithms EsDEr-NR, LSHADE-EpSin, SFS-DP_{DE-GW} and dDSF-EA are ranked at 1st, 3rd, 4th and 6th places, while on 100D, these algorithms ranked 1st, 2nd, 4th and 16th places, respectively. The reason behind the bad performance of SFS-DP_{DE-GW} on 100D is that the update process consumes many function evaluations without improving the search. This issue has been addressed in the enhanced version, dDSF-EA, by introducing a success-based update process which proposes an efficient use of function evaluations given to the update process. This enhancement is clearly shown in [Table 8. 2](#) in which dDSF-EA is ranked 6th compared to 16th place for SFS-DP_{DE-GW}. The results of these tables reveal that these newly developed state-of-the-art DE algorithms were successful, and they provide powerful and successful amalgams. As far as we know, EsDEr-NR is the best performing algorithm in the literature to solve the IEEE CEC'2014 benchmark suite on single objective real-parameter optimization, by the time this thesis was prepared.

Table 8. 1: Average ranking of competitor algorithms, achieved by the Aligned

Friedman test at $D=50$. Best entry is marked in bold

Algorithm	Aligned Friedman Ranking	Order
SinDE	521.7500	22
SaDE	503.2333	21
EPSDE	433.3500	20
Amalgam	398.3167	19
SFS	393.7833	18
OXDE	377.3667	17
JADE	360.8000	16
CoDE	335.8333	15
GaAPADE	331.5667	14
MVMO	316.9000	13
dynNP-jDE	310.8833	12
CoBiDE	309.9167	11
NBIPOP-aCMA-ES	308.8167	10
SHADE	303.7667	9
UMOEAs	295.2167	8
iCMAES-ILS	286.4833	7
SFS-DV _{DE-GW}	283.3167	6
L-SHADE	279.6500	5
dDSF-EA	241.1333	4
LSHADE-EpSin	228.0000	3
UMOEAsII	227.0333	2
EsDEr-NR	223.8833	1
Statistic	25.6195967	
<i>p</i> -value	0.22132648	

Table 8. 2: Average ranking of competitor algorithms, achieved by the Aligned

Friedman test at $D=100$. Best entry is marked in bold

Algorithm	Aligned Friedman Ranking	Order
OXDE	487.9167	22
SaDE	475.2667	21
EPSDE	423.1667	20
SinDE	414.0167	19
SFS	411.5333	18
CoDE	378.7833	17
SFS-DP _{DE-GW}	369.3500	16
JADE	368.8500	15
Amalgam	346.6667	14
CoBiDE	334.6000	13
GaAPADE	311.1167	12
SHADE	303.3500	11
MVMO	301.5333	10
NBIPOP-aCMA-ES	291.4333	9
dynNP-jDE	291.4000	8
L-SHADE	281.0167	7
UMOEAs	274.5500	6
iCMAES-ILS	265.3333	5
dDSF-EA	247.9167	4
UMOEAsII	245.7333	3
LSHADE-EpSin	224.2833	2
EsDEr-NR	223.1833	1
Statistic	26.1249802	
p -value	0.2017235	

PART III

Applications: Improved Differential Evolution Algorithms for Solving Real-World Optimization Problems

Chapter 9 A novel differential crossover strategy based on covariance matrix learning: L- *covn*SHADE Algorithm

9.1. Introduction

The performance of any newly developed evolutionary algorithm is best judged using a set of challenging benchmark functions. Among these benchmark problems, real-world numerical optimization problems are considered the most challenging ones. This is because they include unique and difficult characteristics and constraints such as: complicated objective functions, nonlinear constraints, large number of variables and many local optima within the search space. Adding to their importance that they represent real and important problems from different science and engineering fields. In recent years, many new evolutionary algorithms and differential evolution techniques were proposed to solve real-world optimization problems [\[105\]](#)[\[106\]](#)[\[107\]](#)[\[108\]](#)[\[109\]](#)[\[110\]](#)[\[111\]](#).

Solving real-world optimization problems is considered a challenging task. This is due to the variability of the characteristics in the objective functions, the presence of enormous number of local optima within the search space and highly nonlinear constraints with large number of variables. The advances on this type of problems are of capital importance for many researchers to develop new efficient evolutionary algorithms to tackle such problems in an efficient manner. For this reason, an improved DE algorithm based on a new crossover technique which uses the covariance learning with Euclidean neighbourhood was proposed.

This new crossover is incorporated in the basic L-SHADE algorithm, and the new algorithm is named *L-covnSHADE*. The goal is to help L-SHADE establish a suitable coordinate system for the crossover operator. This helps enhance L-SHADE capability to solve real world problems with difficult characteristics and nonlinear constraints.

The remainder of the chapter is organized as follows. The proposed *L-covnSHADE* is presented in Section 9.1. The experimental set-up and simulation results are presented in Section 9.2. Finally, Section 9.3 summarizes the conclusions of this work.

9.2. *L-covnSHADE*: How it Works

9.2.1. New Crossover Operator

In L-SHADE algorithm, the binomial crossover is the one used at every generation. In the algorithm presented in this chapter and to establish a suitable coordinate system for the crossover operator, the covariance matrix learning with Euclidean neighbourhood is used with a probability denoted as pa . To begin the crossover operator, the individuals are firstly sorted according to their function values and the best individual, X_{best} , is marked. Next, the Euclidean distance is computed between X_{best} and every other individual in the population as presented in [Eq. \(2.11\)](#). The key idea in this approach is to select the individuals which will form the covariance matrix in an effective manner. According to this, the neighbourhood region around the best individual is used to select those individuals. To find the neighbourhood region, the individuals are sorted

according to their Euclidean distances and $NP \times ps$ individuals are chosen around the best individual. ps is the proportion of individuals that are selected in the neighbourhood region, since the population size is dynamically decreased and so $NP \times ps$ is dynamically decreased as well. The covariance matrix, C , is then computed from this neighbourhood region as shown below:

$$C = BD_m B^T \quad (2.12)$$

Where B^T and B are orthogonal matrices and D_m is the diagonal matrix which consists of the Eigen values.

After the covariance matrix is computed, the target and trial vectors are updated using B^T as follows:

$$\bar{X}_{i,g} = B^T X_{i,g} \quad (2.13)$$

$$\bar{V}_{i,g} = B^T V_{i,g} \quad (2.14)$$

The binomial crossover is then applied on $\bar{X}_{i,g}$ and $\bar{V}_{i,g}$ to create a trial vector $\bar{U}_{i,g}$ in the Eigen coordinate system as follows:

$$\bar{u}_{i,g}^j = \begin{cases} \bar{v}_{i,g}^j, & \text{if } (CR < rand) \text{ or } (j = j_{rand}) \\ \bar{x}_{i,g}^j, & \text{otherwise} \end{cases} \quad (2.15)$$

Finally, $\bar{U}_{i,g}$ is transformed into the original coordinate system using matrix B :

$$U_{i,g} = B \bar{U}_{i,g} \quad (2.16)$$

9.2.2. Putting it all together

The new proposed crossover strategy is embedded in L-SHADE algorithm. The overall pseudo-code of L-covnSHADE is presented in [Pseudocode A. 6](#). Lines 1-13 present the original L-SHADE algorithm with initialization and mutation stages with parameter adaptations. After generating the mutant vector, $V_{i,g}$, the covariance matrix learning for the new crossover operator is performed in Lines 14-24 with a pre-defined probability denoted as pa . If a random number, $rand$, is less than pa then the covariance learning is called to perform the crossover as presented in Lines 16-21. Otherwise, the original binomial crossover is performed as shown in Line 23. Next, the new offspring, $U_{i,g}$, is evaluated and the selection operation is started as shown in Lines 25-35. The memory for the control parameters, μF and μCR , is then updated in Line 37. Finally, Lines 39-41 present how to adjust the population size using dynamic reduction.

9.3. Experimental Results

9.3.1. Real-World Benchmark Problems

A challenging set of real-world numerical optimization problems is used to test the performance of L-covnSHADE algorithm. This benchmark set is presented in the IEEE CEC'2011 competition on testing evolutionary algorithms on real-world numerical optimization problems [\[112\]](#). The suite consists of 22 real-world problems referred to as T01–T13 and is summarized in [Table B. 2](#). This benchmark spans different challenging problems from diverse fields such as: sound waves, radar system, network expansion planning, economic dispatch

problems and spacecraft trajectory optimization. In these problems, finding the best solution is a challenging task due to high dimensionality which ranges from 1 to 240, ill-conditioning and multimodality.

9.3.2. Compared Algorithms

The performance of the *L-covn*SHADE algorithm is compared with the performances of other state-of-the-art algorithms as listed below:

1. GA-MPC: genetic algorithm with a new multi-parent crossover. The winner of the IEEE CEC'11 competition [113]
2. SFS: stochastic fractal search [86]
3. EA-DE-MA: performance of a hybrid EA-DE-memetic algorithm [114]
4. SAMODE: differential evolution with multiple strategies [115]
5. L-SHADE: differential evolution with linear population size reduction [40]

The same control parameter values that were suggested in the original papers were used. For the proposed algorithm, the same control settings for L-SHADE algorithm were used, and for the proposed crossover strategy: ps and pa are set to 0.5 and 0.4, respectively. Following the guidelines of the used benchmark set, each algorithm is run for 150,000 function evaluations, and for each problem the algorithm is run 25 times.

9.3.3. Discussion of Results

For the sake of judging the performance of the proposed algorithm, a comparison with the aforementioned algorithms is presented in [Table D. 10](#). In order to facilitate a fair comparison, the population of every algorithm included in the comparison was initialized using a uniform random distribution, following the instructions of the competition [\[112\]](#). The best parametric setups for all the used algorithms were employed as prescribed in their respective sources. In [Table D. 10](#), the comparisons are limited to 150,000 function evaluations and the mean and standard deviation of the achieved function values for the benchmark problems are reported.

A two-sided Wilcoxon rank sum test is used to evaluate the statistical significance of the observed differences between the contestant algorithms [\[87\]](#). The results are summarized as “ $w/t/l$ ” in .

Table 9. 1 and the last row of Table D. 10, to imply that the contestant algorithm wins in w functions, ties in t functions, and loses in l functions, when compared to L-*covn*SHADE. The simulation results show that L-*covn*SHADE performed at least as well as the other contestant algorithms in 19 functions out of 22. Considering function T07, the mean value for L-*covn*SHADE is 1.1223 which is outperformed by EA-DE-MA and SAMODE algorithms in minor difference with mean value (0.5278 and 0.81662, respectively).

Function T09 was the worst case in which L-*covn*SHADE was outperformed by all other used algorithms, and the best results for this function was obtained by EA-DE-MA algorithm with a mean value equal to 5.2214E+02. L-*covn*SHADE got the best results compared to other algorithms in functions: T10.1, T11.4, T11.5, T12.1 and T12.2. By looking at function T13, L-*covn*SHADE has similar performance compared with L-SHADE, while both algorithms are outperformed by all other algorithms. For the last function, T13, L-*covn*SHADE has an equal performance with other algorithms while it outperformed SFS and EA-DE-MA algorithms. A careful scrutiny of .

Table 9. 1 shows that GA-MPC (winner of the used benchmark suite) loses in 13 functions, ties in 7 functions and wins in just 2 functions. Compared to L-SHADE algorithm, L-*covn*SHADE shows superior performance in 7 functions, have an equal performance in 13 functions and loses in just 2 functions. The results reveal the effectiveness of using the proposed crossover startgey to solve a diverse set of real-world optimization problems. L-*covn*SHADE is outperformed by L-SHADE in function T10.2 while both algorithms have the same performance in function T11.3.

Table 9. 1: Comparison of L-*covn*SHADE with state-of-the-art algorithms using the Wilcoxon rank-sum test (significantly, $p < 0.05$)

Vs. L- <i>covn</i> SHADE		
GA-MPC	+ (win)	2
	= (equal)	7
	- (lose)	13
SFS	+ (win)	3
	= (equal)	6
	- (lose)	13
EA-DE-MA	+ (win)	3
	= (equal)	6
	- (lose)	13
SAMODE	+ (win)	2
	= (equal)	6
	- (lose)	14
L-SHADE	+ (win)	2
	= (equal)	13
	- (lose)	7

9.4. Conclusion

This chapter introduces an enhancement of L-SHADE algorithm to solve real-world optimization problems using a new improved crossover strategy. The proposed algorithm is named L-*covn*SHADE in which the covariance matrix learning with Euclidean neighbourhood is used to perform the crossover operator. This approach aims at establishing a better coordinate system where the mutant and target vectors are transformed to Eigen coordinate system before performing the binomial crossover. The covariance matrix which is built from the neighbourhood region of the best individual is used for this transformation. The algorithm is tested on one of the challenging benchmarks which consists of 22 real-world numerical optimization problems from diverse fields. When compared

with other state-of-the-art algorithms including winner of the used benchmark suit and L-SHADE, the proposed algorithm shows good performance.

Chapter 10 An improved differential evolution using efficient adapted surrogate model: iDEaSm

Algorithm

10.1 Introduction

In DE algorithm, like any other evolutionary algorithm, the optimization process continues until the termination criterion is met which is defined as a pre-defined number of function evaluations (FEs). Increasing this number ultimately helps the algorithm to generate better solutions if the algorithm has not converged yet. Real-world optimization benchmarks are subject to many constraints and are often complex in their nature. Typically, such problems are expensive in terms of computational time and cost. Conventional solvers that are used to tackle such problems require a considerable large number of function evaluations. Such budget is not affordable in practice. The algorithm might not converge even after consuming this number, and hence it does not guarantee an optimal solution to be found.

To achieve this, approximations can be used using optimization algorithms to reduce the actual number of FEs [116]. The goal is to approximate the objective function using certain models. Those approximations are referred to as surrogate models which can be used to evaluate the new trial solutions and hence, they are computationally inexpensive when compared to the original objective function [117]. The history of population individuals in an evolutionary algorithm can be used to build a surrogate model which aims to reduce the FEs and hence assist its evolution to generate better solutions [118][119]. Building a surrogate model

involves the use of a function approximation to estimate the response of the original function through training using two sets (X, y) where $X = \{X_1, X_2, \dots, X_N\}$ is a set of vectors (solutions) of D decision variables and $y = \{y_1, y_2, \dots, y_N\}$ is the responses (functions values). Hence, the surrogate model is expressed as $y = f(X)$ which estimates the original objective function.

In this chapter, the main target is not to propose a new differential evolution algorithm, but to propose a new approach to improve its performance and generate new and better solutions using an effective surrogate model. The key advantage of the proposed surrogate model in this work is in the use of a novel scheme to adapt the *theta* parameter. The target is to build an accurate model to approximate the response of an objective function in an effective manner. This adaptation scheme uses the information exploited from the computed covariance matrix of the current population which is then converted to a correlation matrix. This correlation matrix can be used to adapt the *theta* parameter during the search instead of using one fixed value. Using this way, the accuracy of the used surrogate model is enhanced to mimic the response of the original objective function. The proposed surrogate model is used to assist L-SHADE algorithm to produce better solutions by using more than one chance to generate better offspring during the search process. Therefore, the probability to generate competitive offspring solutions is increased. Moreover, the proposed surrogate model is used to enhance the best solution using an auxiliary local search procedure. The new algorithm, an improved differential evolution with an adapted surrogate model, namely iDEaSm, is validated using two different

benchmarks suites. The first one is a set of bound-constrained optimization problems and the second benchmark suite constitutes a set of real-world optimization problems, which is also used to test the validity of the introduced surrogate model to estimate the response on challenging real-life applications. Simulation results show that the proposed algorithm is able to generate good solutions as compared to other algorithms. Also, two well-known engineering design problems are used to test the performance of the proposed algorithm. These problems are the welded beam and the pressure vessel optimization problems. The obtained results demonstrate the effectiveness and the capability of the proposed algorithm in solving different types of problems with variant characteristics and complexity.

The rest of the chapter is organized as follows. Section 10.1 introduces the surrogate model concept and summarizes the use of related surrogate models to assist the performance of DE and other EAs. The proposed algorithm is introduced in Section 10.2. The experimental results and analysis for three experiments are presented in Section 10.3. Finally, the conclusion is summarized in Section 10.4.

10.2 Surrogate Model

A surrogate model is basically a mathematical model and is considered as a statistical model of the response surface of a simulation model. To estimate the behaviour of an objective function, the surrogate model is used which can be defined as a meta-model using the mathematical functions by studying the

relationship between (X, y) where X is the input sample and y is the output response to be presented as:

$$y = f(X) \quad (3.1)$$

The approximation form of the above equation forms the surrogate model as presented below:

$$\hat{y} = \hat{f}(X) \quad (3.2)$$

where the output of the surrogate model mimics the original objective function as shown below:

$$\hat{y} = y + \varepsilon \quad (3.3)$$

Different procedures have been introduced to build the surrogate models such as: support vector machines (SVM) [120], response surface methods (RSM) [121], Kriging [122], artificial neural networks (ANN) [123]. In [124][125], a comparison between the different methods to build a surrogate model is introduced. Those studies used small sets of test problems and considered the use of only one criterion to select the best model. On the other hand, the authors in [119] demonstrated a study on the usage of surrogate models and how to select an appropriate model for the evolutionary algorithms based on two factors which are suitability of combining the surrogate model in an evolutionary algorithm and dimensionality. Based on that, the authors recommend the use of support vector regression method or Kriging model as the best options to estimate the optimization problem with low to moderate dimensions [119]. Moreover, they

concluded that the radial basis function network is the best option to model high dimensionality problems.

The Kriging models have been used in different applications for numerical and engineering problems [122]. It also became popular in the last decade to be used to improve the performance of an evolutionary algorithm to solve many benchmark sets including expensive optimization problems [119]. The function of interest, $y(X)$ in the Kriging model can be expressed as shown in (3.4) using two models which are global and localized to better interpolate the input data:

$$y(X) = f(X) + Z(X) \quad (3.4)$$

The polynomial function, $f(X)$ represents the global definitions of the actual function, and $Z(X)$ is a representation of the localized deviations that represents the stochastic process using a Gaussian model with a mean equals to 0 and variance of σ^2 . The covariance of $Z(X)$ is presented in the following equation:

$$\text{Cov}[Z(X^i), Z(X^j)] = \sigma^2 \mathfrak{R}(\theta, X^i, X^j) \quad (3.5)$$

where $\mathfrak{R}$ is the correlation matrix for a data set of samples. Given two samples, i and j , the correlation between them is represented as $\mathfrak{R}(\theta, X^i, X^j)$.

Many surrogate-assisted differential evolution algorithms have been proposed in the literature [118]. The main goal was to use a well-designed model to mimic the response of an objective function, for the optimization problem being solved, to be used to evaluate the trial vectors. Based on that, the function evaluations using the original objective function are reduced. Most of these algorithms deal

with expensive optimization problems since a limited number of function evaluations is allowed for this type. On the other hand, the surrogate-assisted algorithms have been used to solve practical problems as in [126] and also for multi-objective optimization problems [127]. Sa-DE-DPS algorithm was proposed in [128], where the authors used a simple Kriging model which was built using the initial population samples. Then, a new generation of trial vectors was generated using a different mutation strategy which was selected in a random fashion. Those new trial vectors were then evaluated using the surrogate model. The best individual was selected to be evaluated using the original objective function and then it was added to the samples to rebuild the surrogate model for the next generation. HSBA algorithm was proposed in [129], in which the authors used three stages to solve a benchmark set of expensive optimization problems. The first stage was the global search in which the DE operations were used for 10 generations using the actual objective function. The stored samples in those 10 generations were then used to build a Kriging model to be utilized later to evaluate the trial vectors. In the third stage, the quality of the best solution was enhanced using local search. The authors in [130] used a cheap surrogate model based on density estimation and a multi-operator search strategy. The surrogate model was built using Parzen window method which used the kernel density estimation, and hence the density for each sample point was approximated. ESMDE algorithm was proposed in [131] which was a new version of EPSDE algorithm using an evolving surrogate model. Original EPSDE used an ensemble of different mutation strategies to guide the evolutionary search using a randomly generated number for F and CR .

10.3 iDEaSm Algorithm: How it Works

In this algorithm, the main target is not to propose a new differential evolution algorithm, but to propose a new approach to improve its performance and generate new and better solutions using an effective surrogate model. The key advantage of the proposed surrogate model in this work is in the use of a novel scheme to adapt the *theta* parameter to build an accurate model to approximate the response of an objective function in an effective manner using the Kriging model. This adaptation scheme uses the information exploited from the computed covariance matrix of the current population which is then converted to a correlation matrix. This correlation matrix can be used to adapt the *theta* parameter during the search instead of using one fixed value during the evolutionary search. Using this way, the accuracy of the used surrogate model is enhanced to mimic the response of the original objective function. The proposed surrogate model is used to assist the well-known L-SHADE algorithm [40] to produce better solutions by using more than one chance to generate better offspring during the search process. Therefore, the probability to generate competitive offspring solutions is increased. Moreover, the proposed surrogate model is used to enhance the best solution using an auxiliary local search procedure. The following subsections explain the algorithm in detail.

10.3.1 Proposed Adapted Surrogate Model

In the proposed algorithm, iDEaS, an adapted surrogate model is used to help L-SHADE to generate better solutions during the evolutionary search. The Kriging model is used due to the following reasons [117][122]:

- Its simplicity and capability to estimate the objective function.
- The spatial and temporal correlation of the input data used to estimate the objective function and its shape.
- A good estimation with a satisfactory confidence interval can be obtained with a reasonable computational cost compared to other models.

To build the Kriging model, input samples of $\{X, f(X)\}$ are needed. Those samples are required to use the original function $f(X)$ to evaluate an input X . Then, a mathematical model, as explained in sub-section 10.2, is used to interpolate the samples. As shown in Eq. (3.5), the Kriging model depends mainly on the correlation between the samples, denoted as theta parameter θ , to estimate the response. In the literature, the algorithms which used the Kriging model have used a fixed value for θ , and this value is set in an arbitrary way using one fixed value for all the generations. In this proposed algorithm, an adaptation scheme is used to adapt the correlation over the generations to build a well-designed Kriging model. This scheme falls in the first class which uses the information feedback from the input samples to set the θ parameter, which reflects the actual correlation between the current samples.

The current population at generation g with the function values for its individuals represent the input samples for this Kriging model as shown below:

$$\begin{aligned} &\{X_1, X_2, \dots, X_{NP}\} \\ &\{f(X_1), f(X_2), \dots, f(X_{NP})\} \end{aligned} \tag{3.6}$$

In particular, the extracted information from the population such as covariance, variance and mean values can be utilized to represent its statistical properties.

The information can be extracted by computing the covariance matrix. The key feature that the covariance matrix can provide is the interaction between the variables. Hence, it should be systematically very useful to utilize this matrix to compute the correlation between the variables. Motivated by this consideration, the covariance matrix learning is used in the algorithm presented in this chapter to adapt the θ parameter in the Kriging model. The procedure of θ adaptation is introduced as follows:

Step 1. Use the current population to compute the covariance matrix, and then apply the Eigen decomposition as shown in [Eq. \(3.7\)](#) using the orthogonal matrices: B, B^T and diagonal matrix D_m .

$$C = BD_mB^T \quad (3.7)$$

The Eigen vector of the covariance matrix C represents each column in the matrix B .

Step 2. Compute the square roots of the diagonal variances of C , denoted as the matrix, D_m as shown below:

$$\delta = \sqrt{D_m} \quad (3.8)$$

Step 3. Use the covariance matrix C and δ to compute the correlation matrix E by standardizing each row and column of C as follows:

$$E = \frac{C}{\delta \times \delta'} \quad (3.9)$$

The correlation matrix E contains the correlation coefficients between each variable and the others.

Step 4. Set the θ parameter to the mean value of the upper triangular coefficients of the correlation matrix E as follows:

$$\delta = \text{mean}(\hat{E}) \quad (3.10)$$

Note that, the correlation matrix E is a symmetric matrix and hence the upper triangular coefficients denoted as $\hat{E}$ are taken to compute the mean.

10.3.2 Putting it all together

The proposed adapted surrogate model, which is explained in the previous subsection is incorporated in L-SHADE to help assist the DE evolution to generate better offspring in each generation. A full pseudo-code of iDEaSm is presented in [Pseudocode A. 7](#). The algorithm starts with the initialization step for the used parameter. Next, a population space is created and evaluated. After that, the surrogate model is built after the θ parameter is adapted using the computed coefficient matrix of the current population. Next, the mutation and crossover operations are performed, and the newly generated offspring is evaluated using the surrogate model. If the new offspring is not better than its parent, then another chance is given to generate a new offspring which is evaluated using the surrogate model until the maximum chances are reached, denoted as W .

After the initial population is generated, each individual is evaluated by computing its function value as shown in Lines 4-5 by calling the original function for the problem being solved. Then, at the beginning of each

optimization loop, the proposed surrogate model is constructed using the current population as an input samples denoted as $\{X_1, X_2, \dots, X_{NP}\}$ and $\{f(X_1), f(X_2), \dots, f(X_{NP})\}$. The θ parameter is adapted to build the Kriging model, as explained in subsection 10.3.1, in which the covariance matrix C is computed from the current population, and then it is converted into the correlation matrix E . Next, the mean value of the upper triangular coefficients of this matrix is used to adapt the θ parameter. Then, $y(X)$, denoted as the function of interest, is computed as shown in Eqs. (3.4) - (3.5). Using this adaptation scheme which utilizes the actual interactions between the current samples by using the covariance and correlation matrices help to build a well-designed surrogate model which reflects a good estimation to the original function.

After the surrogate model is built, a new offspring is generated using the mutation and crossover operations of the L-SHADE algorithm. The surrogate model is used to evaluate this new offspring utilizing the function of interest, $y(X)$ which mimics the original function being solved. In case that the new offspring is not better than its parents, different chances are given to generate other offspring individuals using different values for F and CR until a competitive offspring is generated or the maximum number of chances, W , is reached, as shown in Lines 14-24. This way, the proposed surrogate model is utilized to generate good offspring at every generation, and hence assists the evolutionary search to generate competitive solutions. The new competitive offspring values are then evaluated using the original objective function as shown in Lines 25-29.

Replacement occurs in case the new generated offspring has a better function value, and the successful values of F and CR are stored to update the memory as shown in Line 31.

The built surrogate model is also used to enhance the quality of the best solution at each generation using a local search method, based on interior point method as shown in Lines 32-38. Firstly, the best solution is picked and is denoted as $X_{best,g}$. Then, the function of interest $y(X)$ is used by the interior point method to generate a new set of solutions [132]. Each time a new solution is generated by the interior point method, the surrogate model is used to evaluate this solution. Finally, the final solution is evaluated using the original objective function to replace the best solution if it has a better function value. In this manner, the best solution is enhanced by utilizing the surrogate model without using the original function and hence the function evaluations used by this local search are not counted.

At the final stage of iDEaSm, the linear population size reduction is called as shown in Lines 39-43 to dynamically reduce the population size using Eq. (1.27). The individuals are sorted based on their function values, and the worst ranked individuals are eliminated from the population. The surrogate model is re-built using the new population space with its new individuals. This is used to better mimic the actual objective function using the proposed adaptation with the use of the new θ parameter which reflects a new correlation value for the current population. At this point, a new optimization loop is repeated again which utilizes

the new surrogate model to generate new competitive solutions until the termination criterion is met.

10.4 Experimental Results

The performance of iDEaSm is tested using two benchmark sets: bound-constrained benchmark problems and real-world optimization problems. Also, two well-known engineering design problems are used. State-of-the-art algorithms and other evolutionary algorithms are used for comparison. The numerical results and comparisons for the two benchmark sets are presented in the following subsections.

10.4.1 Experiment 1: Bound constrained

Benchmark

In this experiment, 15 bound-constrained problems are tested on 10D and 30D. [Table B. 3](#) in the appendix presents a description of those functions. iDEaSm Matlab 2017a was used for coding. The following state-of-the-art algorithms are used in the comparisons: JADE [\[41\]](#), CoDE [\[51\]](#), SaDE [\[42\]](#), EPSDE [\[50\]](#), ESMDE [\[131\]](#) and L-SHADE [\[40\]](#). The same parametric setting is used for these algorithms as reported in their respective sources. The algorithms used $10,000 \times D$ as the maximum number of function evaluations, and 30 independent runs are used for each problem. In iDEaSm, the same parametric settings for L-SHADE are used in which the initial values for μF and μCR are both set to 0.5 with a memory size equals to 6. $NP_{\min}$ and $NP_{\max}$ are set to 4 and $18 \times D$, respectively, for dynamic reduction. For the proposed surrogate model,

extensive testing revealed that the best number of chances (W) is set to 10. For the local search, any number of function evaluations that the interior point uses can be set as the surrogate model is used to evaluate the new set of generated solutions. In the experiments, 50 function evaluations were used.

The statistical results (mean and standard deviation) of the best final function value for all the algorithms, i.e., $f(\bar{X}_{best})$, $D=10$ and $D=30$ are presented in [Table D. 11](#) and [Table D. 12](#), respectively. To judge whether there is a statistical significance between iDEaSm and every other contestant algorithm, the Wilcoxon test is used [\[87\]](#) with a 0.05 significance level. In these tables, a “+” sign identifies the problems where iDEaSm algorithm exhibits an inferior performance, “=” sign denotes a tie and so a not statistically significant difference and a “-” sign signifies the problem when the compared algorithm fails against the proposed algorithm and hence iDEaSm exhibits a superior performance. The last row of the results tables summarizes the results of Wilcoxon as $w(\text{win}) / t(\text{tie}) / l(\text{lose})$.

The results in [Table D. 11](#) show that the proposed algorithm is at least as good as the other compared algorithms in all the functions. For functions: ($f_1, f_8, f_9, f_{10}, f_{12}$, and f_{13}) iDEaSm shows superior performance and hence the effectiveness of using the proposed adapted surrogate model to assist the evolutionary search to find better solutions. The algorithms have the same mean values for functions ($f_3, f_5, f_6, f_{11}, f_{14}$, and f_{15}). Comparing iDEaSm with L-SHADE algorithm shows that iDEaSm has advanced performance and this demonstrates that the proposed surrogate model with *theta* adaptation has the ability to mimic the original

objective function. Hence, it is able to generate a competitive offspring at each generation. L-SHADE algorithm shows inferior performance in 7 functions, equal performance in 8 functions and does not win in any of the functions.

The performance of the proposed algorithm has not substantially decreased at 30D as shown in [Table D. 12](#). iDEaSm was at least as good as the other algorithms except for f_{12} in which EPSDE algorithm has a better mean value. For functions: (f_3, f_{14} and f_{15}), the compared algorithms have the same mean values. Comparing to L-SHADE algorithm, the proposed algorithm wins in 7 functions and has equal performance in 8 functions. As in 10D, L-SHADE algorithm does not win in any of the functions. This is evident that using this proposed surrogate model enhances the performance of L-SHADE to generate better solutions.

Furthermore, each of the competitor algorithms is ranked using the average Friedman ranking [\[91\]](#). This test is a non-parametric multiple comparison. The test takes an input of n algorithms and performs a statistical analysis to rank each algorithm. The analysis highlights the differences of the performances which are statistically significant of at least one pair of the compared algorithms. [Table 3](#) presents the average Friedman ranking of the competitor algorithms on 10D and 30D. The test-statistic for this test and the corresponding measured p -value are shown in the last two rows of [Table 3](#). The lowest rank is assigned to the best performing algorithm. The results reveal that the proposed algorithm is robust and is able to obtain the best rank which is equal to 2.4 and 2.0 for 10D and 30D, respectively. The second performing algorithm is L-SHADE with a rank equals to 3.2 and 3.4 with a clear difference comparing to iDEaSm on both 10D and 30D, respectively.

Table 10. 1: Average Ranking of competitor algorithms achieved by the Friedman test at dimensions $D=10$ and $D=30$. The best entry is marked in bold.

Algorithm	Ranking ($D=10$)	Ranking ($D=30$)
JADE	5.2667	4.3667
CoDE	4.9000	4.6000
SaDE	3.4333	4.6667
EPSDE	4.1000	3.5000
ESMDE	4.6000	4.6333
L-SHADE	3.2333	3.4333
iDEaSm	2.4667	2.8000
Statistic	19.42857	10.77143
p -value	0.003498	0.095703

10.4.2 Experiment 2: Real-world Optimization

Benchmark

The performance of iDEaSm is also tested on real-world optimization problems to test the validity of the proposed surrogate model to solve this kind of problems. 12 low-dimension problems within the range $6D-40D$ have been taken from the IEEE CEC'11 on real-world numerical optimization problems which are: (T01, T02, T05, T06, T07, T08, T10, T11.2, T11.3, T11.4, T13 and T14) as presented in [Table B. 2](#).

The performance of iDEaSm is compared with the following algorithms including the winner of the IEEE CEC'11 benchmark competition. The same

parametric set-up is used in our algorithm as in experiment 1. Each algorithm is run for 25 independent runs using 50,000 function evaluations.

1. EA-DE-MA: performance of a hybrid EA-DE-memetic algorithm [114]
2. ENSML_DE: ensemble differential evolution algorithm [133]
3. WI_DE: self-adaptive cluster based and weed inspired differential evolution algorithm [134]
4. DE-RHC: Benchmarking a Hybrid DE-RHC Algorithm [135]
5. GA-MPC: GA with a new multi-parent crossover, winner of this competition [113]
6. L-SHADE: Improving the Search Performance of SHADE Using Linear Population Size Reduction [40]

A comparison with the aforementioned algorithms using the mean and standard deviation values at 50,000 function evaluations is presented in [Table D. 13](#) showing the mean and standard deviation for the best objective values for the aforementioned algorithms. The Wilcoxon test is also used to judge the significance of the results compared to iDEaSm algorithm. The last row summarizes the number of functions in which each algorithm shows superior, equal and inferior performances compared to iDEaSm according to Wilcoxon test. The results of [Table D. 13](#) show that iDEaSm is successfully used to solve the challenging real-world problems, and the mean values for the best objective solutions are at least as good as other algorithms for all the problems except for problem T6. DE-RHC algorithm got the best results for this problem with a mean value equal to $-2.8600E+01$. L-SHADE algorithm shows an inferior performance in 9 problems, an equal performance in 3 problems and it never wins in any of

the 12 used real-world problems. This clearly demonstrates that the proposed surrogate model advances the evolutionary search to generate good solutions for real-world problems. Compared to the winner of this competition (GA-MPC), the proposed algorithm wins in 9 problems and shows a tie in 3 problems. The GA-MPC algorithm was not able to beat iDEaSm in any of the used problems. Moreover, the simulation results show the ability of the proposed surrogate model with the adaptation scheme to mimic the response of changing real-world optimization problems. The average Friedman ranking is presented in

Table 10. 2 for each of the used algorithms. The best rank is obtained by iDEaSm and equals to 2.0 compared to the second rank by L-SHADE algorithm which equals to 3.2. The use of such statistical methods demonstrates the effectiveness and robustness of the proposed algorithm.

Table 10. 2: Average Ranking of competitor algorithms achieved by the Friedman test for experiment 2. The best aentry is marked in bold.

Algorithm	Ranking
EA-DE-MA	4.5833
ENSMLE	5.2917
WIDE	3.5417
DE-RHC	4.9167
GA-MPC	4.4167
L-SHADE	3.2500
iDEaSm	2.0000
Statistic	20.04464
<i>p</i> -value	0.002719

10.4.3 Experiment 3: Engineering Design

Optimization Problems

In this section, we study the performance of iDEaSm to solve real-world engineering optimization problems. Two cases studies are proposed in which two well-known engineering optimization problems are used: welded beam design and pressure vessel design problems. The following sub-sections present those case studies and the experimental results of using iDEaSm to solve them with a comparative study.

10.4.3.1 Welded Beam Design Problem

The well-known welded beam design problem is used [136]. The structure of this problem is presented in Figure 10. 1 which consists of a weld that is clamped between two beams: A and B. The optimization task is to find the best dimensions for b , t , h and l which are used to carry a load P and support and generate a minimum fabrication cost.

Figure 10. 1: Framework for the welded beam design problem [136]

The mathematical expression of this problem is expressed as follows which consists of 4 decision variables and 7 constraints to present comprised of the welding labour, material costs, set-up and fabricating cost:

$$\begin{aligned}
f(X) &= 1.10471X_1^2X_2 + 0.04811X_3X_4(14.0 + X_2) \\
\text{subject to,} \\
g_1(X) &= \tau(X) - \tau_{\max} \leq 0, \\
g_2(X) &= \sigma(X) - \sigma_{\max} \leq 0, \\
g_3(X) &= X_1 - X_4 \leq 0, \\
g_4(X) &= 0.10471X_1^2 + 0.04811X_3X_4(14.0 + X_2) - 5.0 \leq 0, \\
g_5(X) &= 0.125 - X_1 \leq 0, \\
g_6(X) &= \delta(X) - \delta_{\max} \leq 0, \\
g_7(X) &= P - P_c(X) \leq 0 \\
\text{where:} \\
\tau(X) &= \sqrt{(\tau')^2 + 2\tau'\tau''\frac{X_2}{2R} + (\tau'')^2}, \\
\tau' &= \frac{P}{\sqrt{2}X_1X_2}, \quad \tau'' = \frac{MR}{J}, \\
M &= P(L + \frac{X_2}{2}), R = \sqrt{\frac{X_2^2}{4} + (\frac{X_1 + X_3}{2})^2}, \\
J &= 2 \left\{ \sqrt{2}X_1X_2 \left[\frac{X_2^2}{12} + (\frac{X_1 + X_3}{2})^2 \right] \right\}, \\
\sigma(X) &= \frac{6PL}{X_4X_3^2}, \quad \delta(X) = \frac{4PL^3}{EX_3^3X_4}, \\
P_c(X) &= \frac{4.013E\sqrt{\frac{X_3^2X_4^6}{36}}}{L^2} \left(1 - \frac{X_3}{2L} \sqrt{\frac{E}{4G}} \right), \\
P &= 6,000\text{lb}, L = 14\text{in}, E = 30 \times 10^6\text{ psi}, G = 12 \times 10^6\text{ psi} \\
\tau_{\max} &= 30,600\text{ psi}, \sigma_{\max} = 30,000, \delta_{\max} = 0.25\text{in}.
\end{aligned} \tag{3.11}$$

Many algorithms have been used to solve this problem including evolutionary algorithms. [Table D. 14](#) presents a comparative study with the best algorithms from literature. In this table, the optimal design variables with the best-found

function costs are presented for the different methods. Also, the table shows the statistical results including best, worst, mean and standard deviation with the number of function evaluations (FEs) which is used to get these values. The results in this table show that iDEaSm is able to generate a similar/better solution among the used methods with the minimum number of function evaluations. The proposed algorithm used 4,425 function evaluations to get a function cost equal to 1.7248. The author in [86] was able to generate the same function cost but with a larger number of function evaluations which equals to 50,000. The results reveal the ability of iDEaSm to solve this problem with high quality solutions compared to other best reported results for other methods.

10.4.3.2 Pressure Vessel Design Problem

The pressure vessel problem is a structural engineering problem consists of a compressed air storage tank with a maximum volume of 75ft^3 and a working pressure of 2000psi [137]. As shown in Fig. 6, the hemispherical heads are capped a cylindrical vessel at both ends. The shell is made in two halves using rolled steel plate and they form a cylinder by joining two longitudinal welds. The objective function is to minimize the total cost which consists of cost of forming, welding and material [137].

Figure 10. 2: Design of pressure vessel problem [137]

Four design variables are associated with this problem: thickness of pressure vessel $T_s (= X_1)$ thickness of head $T_h (= X_2)$, inner radius of the vessel $R (= X_3)$ and length of the vessel without the heads $L (= X_4)$. The mathematical representation of this problem can be expressed as:

$$\begin{aligned}
 f(X) &= 0.6224X_1X_3X_4 + 1.7781X_2X_3^2 + 3.1661X_1X_4 + 19.84X_1^2X_3, \\
 \text{subject to,} \\
 g_1(X) &= -X_1 + 0.0193X_3 \leq 0 \\
 g_2(X) &= -X_2 + 0.00954X_3 \leq 0 \\
 g_3(X) &= -\pi X_3^2X_4 - \frac{4}{3}\pi X_3^3 + 1296000 \leq 0 \\
 g_4(X) &= X_4 - 240 \leq 0 \\
 \text{where: } &0.0625 \leq X_1 \leq 1, 1 \leq X_2 \leq 99, 10 \leq X_{3,4} \leq 200
 \end{aligned} \tag{3.12}$$

Table D. 15 presents a comparison of the best-found solutions for pressure vessel. The results include the optimal design variables, statistical results and the number of FEs to get these results for the cost function. The best function cost is obtained by iDEaSm algorithm which equals to 5988.02732 with 20,000 FEs. The second performing algorithm is the one proposed by [138] and [137] with a cost equal to 6059.71433. The results demonstrate the effectiveness of using the proposed surrogate model to solve such design problem.

10.5 Conclusion

In recent years, evolutionary algorithms attracted the attention of many researchers to develop efficient and new ways to solve a wide spectrum of benchmark problems. Those benchmark problems can be found in almost every science field as optimization problems. Finding this solution with a specified

budget of function evaluations states a challenge for most algorithms. A new improved differential evolution algorithm with a novel adapted surrogate model is introduced in this chapter. In this proposed surrogate model, the covariance matrix, which is utilized to find the coefficient correlations between the parameters, is used to adapt the *theta* parameter in the Kriging model at every generation. The proposed approach falls in the first class which uses an adaptive scheme to control the settings of *theta* parameter for Kriging model. This adaptation helps to mimic the original objective function, and hence assists the evolutionary search to find better solutions. Different chances using different control parameter settings are used to generate high quality offspring in each generation using the proposed surrogate model. The proposed model is embedded in the L-SHADE algorithm. The goal is to advance the search to find better solutions using the surrogate model, and hence not counting the function evaluations. The proposed surrogate model is also used to enhance the quality of the best solution using a local search method that is based on interior-point method. The new algorithm, namely iDEaSm, is validated using two benchmark sets: bound-constrained and real-world optimization problems. The simulation results show that iDEaSm has a statistically good performance and a speedy convergence compared to other state-of-the-art algorithms. Also, the algorithm is tested using two well-known engineering design problems: welded beam and pressure vessel, and the results show the ability of the proposed algorithm to solve such kind of problems comparing to other best methods from the literature.

Chapter 11 Conclusions and Recommendations

11.1 Conclusions

In recent years, evolutionary algorithms have become well-known in almost every science and engineering field for providing good solutions for optimization problems when common mathematical methods fail to do so or when these methods need extra computational cost. Many real-world problems are considered optimization problems because of their nature that one or more objective functions need to be optimized. Such problems state a challenge for many researchers to design efficient algorithms that are capable to provide high quality solutions. As a result, many improved evolutionary algorithms have been introduced and developed in the literature. This thesis has covered one of the most powerful evolutionary algorithms, which is differential evolution. The primary focus is to introduce new differential evolution variants to solve a diverse set of optimization problems efficiently compared to other algorithms from the literature. Efforts are made to propose novel ideas which help the differential evolution algorithm to produce high quality solutions, and hence provide better solutions compared to the best results in the literature.

The thesis is organized in three main parts. In each part, new improved differential evolution algorithms are proposed to address an open issue successfully. In Part I, the focus was on the hybridization concept to enhance the evolutionary search of differential evolution. Two algorithms are introduced in this part: SFS-DP_{DE-GW} and *d*DSF-EA which are discussed in Chapters 3 and 4, respectively. In these two algorithms, the differential evolution is hybridized with

the stochastic fractal search algorithm. SFS-DP_{DE-GW} merges the use of SHADE mutation and Gaussian walks in the diffusion process of stochastic fractal search. dDSF-EA is an enhanced version of SFS-DP_{DE-GW} in which a success-based scheme is used to synchronise the call of two update processes of stochastic fractal search. The two algorithms are tested on the latest developed benchmark suite in that time which is taken from IEEE-CEC'14 competition on real parameter single objective optimization. A summary with a comparison among the state-of-the-art algorithms is presented in Chapter 5. The results are validated using statistical tests including Wilcoxon rank sum test and Friedman ranking which show that the two proposed algorithms have a strong potential to solve this suite efficiently compared to other state-of-the algorithms from literature.

The effort is continued in Part II which focuses on introducing new novel self-adaptive methods to control the settings of differential evolution. It is well-known that the performance of differential evolution is highly dependent on control parameter settings. Hence, this part has covered this open issue by introducing two improved algorithms which are LSHADE-EpSin and EsDEr-NR as presented in Chapters 6 and 7, respectively. In these two algorithms, a new ensemble sinusoidal parameter adaptation is proposed to control the settings of differential evolution. This ensemble pool uses two different sinusoidal adjustments and Cauchy distribution to control the settings of the scaling factor in an efficient manner. This helps to balance the exploration and exploitation phases of differential evolution. EsDEr-NR enhances the performance of LSHADE-EpSin by introducing a new novel idea to control the settings of the population size using a niching-based reduction scheme. In this technique, the

Euclidean distance plays the dominant role to reduce the size for the next generation. In Chapter 8, a summary for this part with a comprehensive comparison is presented including the two developed algorithms in Part I and many other state-of-the-art algorithms. The statistical analysis validates that the best performing algorithm to solve IEEE-CEC'14 benchmark suite is EsDEr-NR algorithm. This because of its novel adaptive settings which control DE parameters effectively.

In Part III, the main target is to propose new efficient differential evolution algorithms to solve real-world optimization applications. Chapter 9 introduces *L-covnSHADE* algorithm in which an improved crossover strategy is proposed. This new strategy uses the covariance matrix learning to build a suitable coordinate system for crossover strategy in differential evolution. The algorithm is introduced to solve diverse set of real-world optimization problems from different fields such as radar system, dynamic and static economic dispatch, hydrothermal scheduling and spacecraft trajectory optimization. The results show that the proposed algorithm solves these problems efficiently.

In Chapter 10, *iDEaSm* algorithm is introduced which conducts the use of a well-designed surrogate model to assist the performance of differential evolution to generate better solutions. This algorithm is the first algorithm which proposes a novel adaptation scheme to control the settings of the Kriging model to better estimate the response of the objective function to be solved. The main target is to use this estimation to evaluate the newly generated offspring and hence reduce the number of used function evaluations. The algorithm is tested in different benchmark suites including real-world problems and engineering design

problems. The results affirm the efficiency of the proposed algorithm to produce high quality solutions compared to results from literature.

11.2 Recommendations for Future Work

This thesis has presented several improved differential evolution algorithms to solve a diverse set of optimization problems. For future work, additional enhancements can be made. One of them is to improve the performance of *L-covnSHADE* algorithm to solve the IEEE CEC'11 in a better way. In this benchmark, the last two problems are spacecraft trajectory optimization problems which are: Messenger and Cassini. Those two problems are taken from the European Space Agency site (ESA) and in their database. The best online results for those problems are reported on their website. The main goal will be to provide better solutions that are competitive enough to break the record, and to test other trajectory optimization problems as well.

Moreover, the aim will be at improving the performance of *EsDEr-NR* algorithm to solve the recently developed benchmark suite on single objective optimization problems. An idea of dividing the population space into different sub-populations with an ensemble of control parameter settings will be investigated. The goal is to check whether all the sub-populations are taking the same evolutionary path or not, and to control their directions well using ensemble adaptations and quality-controlled amalgams.

Furthermore, the use of surrogate methods with ensemble mutation strategies and local searches will be conducted to solve computationally expensive optimization problems, and other well-known constrained engineering design optimization. In

these problems, the number of function evaluations is limited, and the goal is to use a well-designed surrogate method to approximate the response of the tested function. Hence, the surrogate method will be cheaper than the original function and this will enhance the performance of the proposed algorithm to generate new better solutions.

Another potential direction is to introduce new differential evolution algorithms to solve power engineering problems. Also, one of the future plans is to conduct the use of differential evolution for the control and navigation problem of unmanned aerial vehicles. A new version of differential evolution will be proposed to train the neural networks instead of traditional methods. The target is to develop enhanced novel methods to control UAVs for different purposes such as: path planning, trajectory tracking, path following, parameter identification for multi-rotor UAVs and offline and online training. More science and engineering problems will be considered, and the goal will be to provide competitive, cost effective, and efficient ways of search.

AUTHOR'S PUBLICATIONS

Journal papers

1. **Noor H. Awad**, Mostafa Z. Ali, Rammohan Mallipeddi, Ponnuthurai N. Suganthan, An Improved Differential Evolution Algorithm using Efficient Adapted Surrogate Model for Numerical Optimization. Inf. Sci, vols. 451–452, pp. 326-347, 2018.
2. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Ensemble of parameters in a sinusoidal differential evolution with niching-based population reduction, Swarm and Evolutionary Computation, vol. 39, pp. 141-156, 2017.
3. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Edward Jaser, A decremental stochastic fractal differential evolution for global numerical optimization, Inf. Sci., vol. 372, pp. 470-491, 2016.
4. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, An Efficient Differential Evolution Algorithm for Stochastic OPF based Active-Reactive Power Dispatch Problem considering Renewable Generators, applied soft computing, accepted.
5. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, A New Damped Sine Differential Evolution Algorithm for Global Numerical Optimization, applied intelligence, submitted

Conference papers

6. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Robert G. Reynolds, Ali M. Shatnawi, A novel differential crossover strategy based on covariance matrix learning with Euclidean neighborhood for solving real-world problems, in Proc. IEEE Congr. Evol. Comput., CEC 2017, pp. 380-386
7. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Robert G. Reynolds, An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CEC2014 benchmark problems, in Proc. IEEE Congr. Evol. Comput., CEC 2016, pp. 2958-2965.
8. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Edward Jaser, Differential evolution with stochastic fractal search algorithm for global numerical optimization, in Proc. IEEE Congr. Evol. Comput., CEC 2016, pp. 3154-3161.
9. **Noor H. Awad**, Mostafa Z. Ali, Ponnuthurai N. Suganthan, Ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood for solving CEC2017 benchmark problems, in Proc. IEEE Congr. Evol. Comput., CEC 2017, pp. 372-379.
10. Vikas Palakonda, **Noor H. Awad**, Rammohan Mallipeddi, Mostafa Z. Ali, Suganthan Ponnuthurai Nagaratnam, Differential Evolution with Stochastic Selection for Uncertain Environments: A Smart Grid Application. Accepted in in Proc. IEEE Congr. Evol. Comput., Rio Brazil, July 2018. Shortlisted for the Best Student Paper Award - IEEE CEC 2018

11. Partha P. Biswas, **Noor H. Awad**, Ponnuthurai N. Suganthan, Mostafa Z. Ali, Gehan A. Amaratunga, Minimizing THD of multilevel inverters with optimal values of DC voltages and switching angles using LSHADE-EpSin algorithm, in Proc. IEEE Congr. Evol. Comput., CEC 2017, pp. 77-82, 2017

Technical Reports

12. **Noor H. Awad**, Mostafa Z. Ali, J.J. Liang, B.Y. Qu, Ponnuthurai N. Suganthan, “Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization,” Nanyang Technological University, Jordan University of Science and Technology and Zhengzhou University, Tech. Rep., 2016.

BIBLIOGRAPHY

- [1] S. Das, P. Suganthan, Differential evolution: A survey of the state-of-the-art, *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 4–31, Feb. 2011.
- [2] S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution - An updated survey, *Swarm and Evolutionary Computation*, vol. 27, pp. 1-30, 2016.
- [3] K.V. Price, R.M. Storn, J.A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*, 1st ed. New York: Springer-Verlag, Dec. 2005.
- [4] R. Storn, K. Price, Differential evolution a simple and efficient heuristic for global optimization over continuous spaces, *J. Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
- [5] R. Gamperle, S. D. Muller, P. Koumoutsakos, A parameter study for differential evolution, in *Proc. WSEAS Int. Conf. Adv. Intell. Syst. Fuzzy Syst. Evol. Comput.*, Vancouver, BC, Canada, pp. 293–298, 2002.
- [6] J. Liu, J. Lampinen, On setting the control parameter of the differential evolution method, in *Proc. 8th Int. Conf. Soft Comput.*, Brno, Czech Republic, pp. 11–18, 2002.
- [7] E. Mezura-Montes, J. Velazquez-Reyes, C.A. Coello Coello, A comparative study of differential evolution variants for global optimization, in *Proc. 8th Annu. Conf. Genet. Evol. Comput. (GECCO)*, New York, NY, USA, pp. 485–492, 2006.

- [8] F. Neri, V. Tirronen, Recent advances in differential evolution: A survey and experimental analysis, *Artif. Intell. Rev.*, vol. 33, nos. 1–2, pp. 61–106, Feb. 2010.
- [9] J. Lampinen, I. Zelinka, On stagnation of the differential evolution algorithm, in *Proc. 6th Int. Mendel Conf. Soft Comput.*, Shenyang, China, pp. 76–83, 2000.
- [10] N. Padhye, P. Mittal, K. Deb, Differential evolution: Performances and analyses, in *Proc. IEEE Congr. Evol. Comput.*, Cancun, Mexico, pp. 1960–1967, 2013.
- [11] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, Technical Report, Nanyang Technological University, Singapore, May 2005 and KanGAL Report, IIT Kanpur, India, 2005.
- [12] J.J. Liang, B.Y. Qu, P.N. Suganthan, A.G. Hernandez-Diaz, Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session and Competition on Real-Parameter Optimization, Technical Report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, January 2013.
- [13] J.J. Liang, B.Y. Qu, P.N. Suganthan, Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Real-Parameter Optimization, Technical Report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, January 2014.

- [14] N.H. Awad, M.Z. Ali, J.J. Liang, B.Y. Qu, P. N. Suganthan, Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization, Nanyang Technological University, Jordan University of Science and Technology and Zhengzhou University, Tech. Rep., 2016. C
- [15] T. Back, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford, U.K.: Oxford Univ. Press, 1996.
- [16] P.A. Vikhar, Evolutionary algorithms: A critical review and its future prospects., in Proc. of the 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC). Jalgaon, pp. 261-265, 2016.
- [17] D. Ashlock, Evolutionary Computation for Modeling and Optimization, Springer, 2006.
- [18] T. Back, D. Fogel, Z. Michalewicz, Handbook of Evolutionary Computation, Oxford Univ. Press, 1997.
- [19] Z. Michalewicz, D.B. Fogel, How To Solve It: Modern Heuristics, Springer, 2004.
- [20] D. Simon, Evolutionary Optimization Algorithms, Wiley, 2013. C2.64
- [21] W. Banzhaf, P. Nordin, R. Keller, F. Francone, Genetic Programming - An Introduction, Morgan Kaufmann, San Francisco, 1998.
- [22] P.E. Gill, W. Murray, Quasi-newton methods for unconstrained optimization, IMA. J. Appl. Math., vol. 9, pp. 91–108, 1986.
- [23] F. Glover, Tabu search-part I, ORSA J. Comput., vol. 1, pp. 190–206, 1989.

- [24] F. Glover, Future paths for integer programming and links to artificial intelligence, *Comput. Oper. Res.*, vol. 13, pp. 533–549, 1972.
- [25] A. Nelder, R. Mead, A simplex method for function optimization, *Comput. J.*, vol. 7, pp. 308–313, 1965.
- [26] N. Shor, *Minimization methods for non-differentiable function*, Springer-Verlag, Berlin, 1985
- [27] H. Schramm, J.A. Zowe, A version of the bundle idea for minimizing a nonsmooth function: conceptual idea, convergence analysis, numerical results, *SIAM J. Optim.* 2 (1) (1992) 121–152
- [28] P.M. Vasant, *Meta-Heuristics Optimization Algorithms in Engineering, Business, Economics, and Finance*, IGI Global, 2012.
- [29] P. Vasant, G.-W. Weber, V.N. Dieu, *Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics*, IGI Global, 2016.
- [30] A.M. Anile, V. Cutello, G. Nicosia, R. Rascuna, S. Spinella, Comparison among Evolutionary Algorithms and Classical Optimization Methods for Circuit Design Problems, in *Proc. IEEE Conference on Evolutionary Computation*, vol. 1 Vancouver, Canada, pp. 765-772, 2005.
- [31] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, Chichester, England: John Wiley & Sons, 2002.
- [32] J.H. Holland, *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.
- [33] J. Kennedy, R. Eberhart, Particle swarm optimization, in *Proc. of IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

- [34] M. Dorigoet, L.M. Gambardella, Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Trans. Evol. Comput.*, vol. 1, pp. 53-66, 1997.
- [35] D.D. Karaboga, An Idea Based On Honey Bee Swarm for Numerical Optimization, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
- [36] R.G. Reynolds, An introduction to cultural algorithms, in: *Proc. Annu. Conf. Evol. Program.*, pp. 131–139, 1994.
- [37] Hutter, H. Hoos, K. Leyton-Brown, Sequential Model-Based Optimization for General Algorithm Configuration, in *Proc of learning and intelligent optimization, LION-5*, 2011
- [38] M.N. Katehakis, A.F. Veinott, The Multi-Armed Bandit Problem: Decomposition and Computation, *Mathematics of Operations Research*, vol. 12, no. 2, pp. 262–268, 1987
- [39] R. Tanabe, A. Fukunaga, Success-History Based Parameter Adaptation for Differential Evolution, in *IEEE CEC*, 2013, pp. 71–78.
- [40] R. Tanabe, A. Fukunaga, “Improving the Search Performance of SHADE Using Linear Population Size Reduction,” in *Proc. IEEE Congr. Evol. Comput.*, 2014, pp. 1658 - 1665.
- [41] J. Zhang, A.C. Sanderson, JADE: Adaptive differential evolution with optional external archive, *IEEE Trans Evol Comput*, vol. 13, no. 5, pp. 945–958, 2009.
- [42] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization, *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, April, 2009.

- [43] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Natural Computing. New York: Springer, 2003
- [44] J. Tvrdik, I. Krivy, L. Misik, Evolutionary algorithm with competing heuristics, in Proc. MENDEL 2001, Int. Conf. Soft Computing, Brno, Czech, pp. 58–64, Jun. 2001.
- [45] J.R. McDonnell, R.G. Reynolds, D.B. Fogel, Adapting crossover in evolutionary algorithms,”MIT press, The 4th Annual Conference on Evolutionary Programming, MIT Press, pp. 367–384, 1995
- [46] V. L. Huang, A. K. Qin, P. N. Suganthan, Self-adaptive differential evolution algorithm for constrained real-parameter optimization, in Proc. IEEE Congr. Evol. Comput., pp. 17–24, Jul. 2006.
- [47] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE Trans. Evol. Comput., vol. 10, no. 6, pp. 646–657, Dec. 2006.
- [48] J. Brest, B. Boskovic, S. Greiner, V. Zumer, M. S. Maucec, Performance comparison of self-adaptive and adaptive differential evolution algorithms, Soft Comput.: Fusion Found., Methodologies Applicat., vol. 11, no. 7, pp. 617–629, 2007.
- [49] J. Brest and M. S. Maucec, Population size reduction for the differential evolution algorithm, Appl. Intell., vol. 29, no. 3, pp. 228–247, 2008.
- [50] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, M. F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput., vol. 11, no. 2, pp. 1679–1696, 2011.

- [51] Y. Wang, Z. Cai, Q. Zhang, Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters, *IEEE Tran. Evol. Comput.*, vol. 15, no. 1, pp. 55–66, 2011.
- [52] M. Yang, J. Guan, L. Changhe, Differential Evolution with Auto enhanced Population Diversity: the Experiments on the CEC'2016 Competition, in *Proc. IEEE Congr. Evol. Comput.*, 2016.
- [53] A. Viktorin, M. Pluhacek, R. Senkerik, Success History Based Adaptive Differential Evolution Algorithm with Multi Chaotic Framework for Parent Selection Performance, in *Proc. IEEE Congr. Evol. Comput.*, 2016.
- [54] P. Bujok, J. Tvrdek, R. Polakova, Evaluating the Performance of SHADE with Competing Strategies on CEC 2014 Single Parameter Test Suite, in *Proc. IEEE Congr. Evol. Comput.*, 2016.
- [55] http://www.ntu.edu.sg/home/epnsugan/index_files/cec-benchmarking.htm
- [56] R. Tanabe, A. Fukunaga, Tuning Differential Evolution for Cheap, Medium, and Expensive Computational Budgets, in *Proc. IEEE Congr. Evol. Comput.*, pp. 2018-2025, 2015.
- [57] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in *IEEE CEC*, pp. 1769–1776, 2005.
- [58] N. Hansen, Benchmarking a BI-population CMA-ES on the BBOB- 2009 function testbed, in *GECCO (Companion)*, pp. 2389–2396, 2009.
- [59] T. Liao, M.A. M. Oca, T. Stutzle, Computational results for an automatically tuned CMA-ES with increasing population size on the cec'05 benchmark set, *Soft Comput.*, vol. 17, no. 6, pp. 1031–1046, 2013.

- [60] K.M. Sallam, R.A. Sarker, D.L. Essam, S.M. Elsayed, Neurodynamic differential evolution algorithm and solving CEC2015 competition problems, in Proc. IEEE Congr. Evol. Comput., 2015.
- [61] N.H. Awad, M.Z. Ali, R.G. Reynolds, A Differential Evolution Algorithm with Success based Parameter Adaptation for CEC2015 Learning based Optimization, in Proc. IEEE Congr. Evol. Comput., 2015.
- [62] J. Brest, M.S. Maucec, B. Boskovic, Improved LSHADE Algorithm for Single Objective Real Parameter Optimization, in Proc. IEEE Congr. Evol. Comput., 2016.
- [63] R. Polakova, J. Tvrdik, P. Bujok, Evaluating the Performance of LSHADE with Competing Strategies on CEC2014 Single Parameteroperator Test Suite, in Proc. IEEE Congr. Evol. Comput., 2016.
- [64] R. Polakova, J. Tvrdik, P. Bujok, LSHADE with Competing Strategies Applied to CEC2015 Learning based Test Suite, in Proc. IEEE Congr. Evol. Comput., 2016.
- [65] J. Brest, M.S. Maucec, B. Boskovic, Single objective real-parameter optimization: Algorithm JSO, in Proc. IEEE Congr. Evol. Comput., 2017.
- [66] A.W. Mohamed, A.A. Hadi, A.M. Fattouh, K.M. Jambi, LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems, in Proc. IEEE Congr. Evol. Comput., 2017.
- [67] A. Zamuda, Adaptive Constraint Handling and Success History Differential Evolution for CEC 2017 Constrained Real-Parameter Optimization, in Proc. IEEE Congr. Evol. Comput., 2017.
- [68] S.M. Guo, J.S.H. Tsai, C.C. Yang, P.H. Hsu, A self-optimization approach for L-SHADE incorporated with eigenvector-based crossover

- and successful-parent-selecting framework on CEC 2015 benchmark set, in Proc. IEEE Congr. Evol. Comput., pp. 1003 – 1010, 2015.
- [69] M. Yang, C. Li, Z. Cai, J. Guan, Differential Evolution with Auto-Enhanced Population Diversity, IEEE Trans. Cybernetics, vol. 45, pp. 302 – 315, 2015.
 - [70] Y.-L. Li, Z.-H. Zhan, Y.-J. Gong, W.-N. Chen, J. Zhang, Y. Li, Differential Evolution with an Evolution Path: A DEEP Evolutionary Algorithm, IEEE Trans. Cybernetics, vol. 45, pp. 1798 – 1810, 2015.
 - [71] S.-M. Guo, C.-C. Yang, P.-H. Hsu, J. S.-H. Tsai, Improving Differential Evolution With a Successful-Parent-Selecting Framework, IEEE Trans. Cybernetics, vol. 19, pp. 717 – 730, 2015.
 - [72] W. Gong, Z. Cai, D. Liang, Adaptive Ranking Mutation Operator Based Differential Evolution for Constrained Optimization, IEEE Trans. Cybernetics, vol. 45, pp. 716-727, 2015.
 - [73] A. Zamuda, J. Brest. Self-adaptive control parameters' randomization frequency and propagations in differential evolution, Swarm and Evolutionary Computation, vol. 25, pp. 72-99, 2015.
 - [74] S. Hui, P.N. Suganthan, Ensemble and Arithmetic Recombination-Based Speciation Differential Evolution for Multimodal Optimization, IEEE Trans. Cybernetics, vol. 46, no. 1, pp. 64-74, 2016.
 - [75] L. Zheng, S.X. Zhang, S.Y. Zheng, Y.M. Pan, Differential Evolution Algorithm With Two-Step Subpopulation Strategy and Its Application in Microwave Circuit Designs, IEEE Trans. on Industrial Informatics, vol. 12, no. 3, pp. 911-923, June 2016.

- [76] Z. Gao, Z. Pan, J. Gao, Multimutation Differential Evolution Algorithm and Its Application to Seismic Inversion, *IEEE Trans. on Geoscience and remote sensing*, vol. 54, no. 6, pp. 3626-3636, June 2016.
- [77] M.Z. Ali, N.H. Awad, P.N. Suganthan, R.G. Reynolds, An Adaptive Multipopulation Differential Evolution With Dynamic Population Reduction, *IEEE Transactions on Cybernetics*, Vol. pp, Issue, 99, Ocb. 2016.
- [78] Y. Wang, B. Xu, G. Sun, S. Yang, A Two-Phase Differential Evolution for Uniform Designs in Constrained Experimental Domains, *IEEE Trans. Evol. Comput.*, vol. 21, Issue, 5, Oct. 2017.
- [79] N.R. Sabar, J. Abawajy, J. Yearwood, Heterogeneous Cooperative Co-Evolution Memetic Differential Evolution Algorithm for Big Data Optimization Problems, *IEEE Trans. Evol. Comput.*, vol. 21, issue. 2, April. 2017
- [80] S. Dominguez-Isidro, E. Mezura-Montes, A cost-benefit local search coordination in multimeme differential evolution for constrained numerical optimization problems, *Swarm and Evolutionary Computation*, vol. 39, pp. 249-266, April 2018.
- [81] K. Opara, J. Arabas, Comparison of mutation strategies in Differential Evolution – A probabilistic perspective, *Swarm and Evolutionary Computation*, vol. 39, pp. 53-69, April. 2018.
- [82] A.P. Piotrowski, J.J. Napiorkowski, Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure?, *Swarm and Evolutionary Computation*, In press accepted manuscript, April. 2018.

- [83] R.D. Al-Dabbagh, F. Neri, N. Idris, M.S. Baba, Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy, *Swarm and Evolutionary Computation*, In press accepted manuscript, April. 2018.
- [84] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-Adaptive Multimethod Search for Global Optimization in Real-Parameter Spaces, *IEEE Trans. Evol. Comput.*, vol. 13, pp. 243-259, 2009.
- [85] S.M. Elsayed, R.A. Sarker, D.L. Essam, N.M. Hamza, Testing united multi-operator evolutionary algorithms on the CEC2014 real-parameter numerical optimization, in: *Proc. IEEE Congr. Evol. Comput.*, pp. 1650-1657, 2014.
- [86] H. Salimi, Stochastic fractal search: a powerful metaheuristic algorithm, *Knowl. Based Syst.*, vol. 75, pp. 1–18, 2015.
- [87] F. Wilcoxon, Individual comparisons by ranking methods, *Biometrics*, vol. 1, pp. 80-83, 1945.
- [88] J.L.J. Laredo, C. Fernandes, J.J.M. Guervos, C. Gagne, Improving Genetic Algorithms Performance via Deterministic Population Shrinkage, in *Proc. GECCO*, pp. 819–826, 2009.
- [89] Y. Wang, Z. Cai, Q. Zhang, Enhancing the search ability of differential evolution through orthogonal crossover. *Inf. Sci.*, vol. 185, pp. 153-177, 2012.
- [90] Y. Wang, H.-X. Li, T. Huang, L.Li, Differential evolution based on covariance matrix learning and bimodal distribution parameter setting, *Applied Soft Computing*, vol. 18, pp. 232–247, 2014.

- [91] J. Demsar, Statistical comparisons of classifiers over multiple data sets, *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.
- [92] J. Derrac, S. Garcia, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, *Swarm and Evolutionary Computation*, vol. 1, pp. 3–18, 2011.
- [93] F.G. Lobo, C.F. Lima, A Review of Adaptive Population Sizing Schemes in Genetic Algorithms, in *Proc. Genet. Evol. Comput., GECCO*, pp. 228–234, 2005.
- [94] J. Arabas, Z. Michalewicz, and J. J. Mulawka, “GAVaPS - A Genetic Algorithm with Varying Population Size,” in *ICEC*, 1994, pp. 73–78.
- [95] C. Garcia-Martinez, M. Lozano, F. Herrera, D. Molina, A.M. Sanchez, Global and local real-coded genetic algorithms based on parent-centric crossover operators, *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1088–1113, 2008.
- [96] M.A.M. de Oca, T. Stutzle, K.V. Enden, M. Dorigo, Incremental Social Learning in Particle Swarms, *IEEE Trans. on Sys., Man, and Cyber., PartB*, vol. 41, no. 2, pp. 368–384, 2011.
- [97] J.L.J. Laredo, C. Fernandes, J.J.M. Guervos, C. Gagne, Improving Genetic Algorithms Performance via Deterministic Population Shrinkage, in *Proc. Genet. Evol. Comput., GECCO*, pp. 819–826, 2009.
- [98] J. Teo, Differential evolution with self-adaptive populations, in *Proc. 9th Int. Conf. Knowl.-Based Intell. Inf. Eng. Syst. (KES)*, Melbourne, VIC, Australia, pp. 1284–1290, 2005.

- [99] A. Draa, S. Bouzoubia, I. Boukhalfa, A sinusoidal differential evolution algorithm for numerical optimisation, *Applied Soft Computing*, vol. 27, 2015, pp. 99-126.
- [100] R. Mallipeddi, G. Wu, M. Lee, P.N. Suganthan, Gaussian Adaptation based Parameter Adaptation for Differential Evolution, in *Proc. IEEE Congr. Evol. Comput.*, pp. 1760 – 1767, 2014.
- [101] I. Erlich, J.L. Rueda, S. Wildenhues, F. Shewarega, Evaluating the Mean-Variance Mapping Optimization on the IEEE-CEC 2014 Test Suite, in *Proc. IEEE Congr. Evol. Comput.*, pp. 1625 – 1632, 2014.
- [102] S.M. Elsayed, N.M. Hamza, R.A. Sarker, Testing United Multi operator Evolutionary Algorithms-II on Single Objective Optimization Problems, in *Proc. IEEE Congr. Evol. Comput.*, 2016.
- [103] I. Loshchilov, CMA-ES with Restarts for Solving CEC 2013 Benchmark Problems, in *Proc. IEEE Congr. Evol. Comput.*, CEC, pp. 369–376, 2013.
- [104] T. Liao, T. Stutzle, Benchmark Results for a Simple Hybrid Algorithm on the CEC 2013 Benchmark Set for Real-parameter Optimization, in *Proc. IEEE Congr. Evol. Comput.*, CEC, pp. 1938–1944, 2013.
- [105] A. Abdullah, S. Deris, S. Anwar, S.N.V. Arjunan, An evolutionary firefly algorithm for the estimation of nonlinear biological model parameters, *PLoS One*, vol. 8, no. 3, 2013.
- [106] Boussaid, A. Chatterjee, P. Siarry, M. Ahmed-Nacer, Hybridizing biogeography-based optimization with differential evolution for optimal power allocation in wireless sensor networks, *IEEE Trans. Veh. Technol.*, vol. 60, no. 5, pp. 2347–2353, 2011.

- [107] T. Chakraborti, A. Chatterjee, A. Halder, A. Konar, Automated emotion recognition employing a novel modified binary quantum-behaved gravitational search algorithm with differential mutation, *Expert Syst.*, vol. 32, no. 4, pp. 522–530, 2015.
- [108] M.Z. Ali, P.N. Suganthan, R.G. Reynolds, A.F. Al-Badarnah, Leveraged Neighborhood Restructuring in Cultural Algorithms for Solving Real-World Numerical Optimization Problems, *IEEE Trans. Evol. Comp.*, vol. 20, no. 2, pp. 218-231, 2016.
- [109] X. Qiu, J.-X. Xu, Y. Xu, K.C. Tan, A New Differential Evolution Algorithm for Minimax Optimization in Robust Design, *IEEE Transactions on Cybernetics*, vol. 48, issue 5, May 2018.
- [110] A. Mohammadi, M. Saraee, Finding influential users for different time bounds in social networks using multi-objective optimization, *Swarm and Evolutionary Computation*, vol. 40, pp. 158-165, June 2018.
- [111] M.R. Maheri, M. Talezadeh, An Enhanced Imperialist Competitive Algorithm for optimum design of skeletal structures, *Swarm and Evolutionary Computation*, vol. 40, pp. 24-36, June 2018.
- [112] S. Das, P. N. Suganthan, Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems, Dept. Electron. Telecommun. Eng., Jadavpur Univ., Kolkata, India, Nanyang Technological Univ., Singapore, Tech. Rep., Dec. 2010.
- [113] M.S. Elsayed, A.R. Sarker, D.L. Essam, GA with a New Multi-Parent Crossover for Solving IEEE-CEC2011 Competition Problems, CEC 2011

- Evolutionary Computation (CEC), in Proc. IEEE Congr. Evol. Comput., pp. 1034 – 1040, 2011.
- [114] H.K. Singh, T. Ray, Performance of a Hybrid EA-DE-Memetic Algorithm on CEC 2011 Real World Optimization Problems, in Proc. IEEE Congr. Evol. Comput., pp. 1322 - 1326, New Orleans, June 2011.
 - [115] S. Elsayed, R. Sarker, D. Essam, Differential Evolution with Multiple Strategies for Solving CEC2011 Real-world Numerical Optimization Problems”, in Proc. IEEE Congr. Evol. Comput., pp. 1041 - 1048, New Orleans, June 2011.
 - [116] A. Ratle, Accelerating the convergence of evolutionary algorithms by fitness landscape approximation, in Parallel Problem Solving from Nature - PPSN V, ser. Lecture Notes in Computer Science, vol. 1498, pp. 87–96, 1998.
 - [117] D. Buche, N.N. Schraudolph, P. Koumoutsakos, Accelerating evolutionary algorithms using fitness function models, in Proceedings of GECCO Workshop on Learning, Adaptation and Approximation in Evolutionary Computation, pp. 166–169, 2003.
 - [118] Y. Jin, A comprehensive survey of fitness approximation in evolutionary computation, *Soft Comput.*, vol. 9, pp. 3–12, 2003.
 - [119] A. Diaz-Manriquez, G. Toscano-Pulido, W. Gomez-Flores, on the selection of surrogate models in the evolutionary optimization algorithms, in Proc. IEEE Congr. Evol. Comput., pp. 2155–2162, 2011.
 - [120] S.R. Gunn, Support vector machines for classification regression, Technical report, school of electronics and computer science, university of southampton, 1998.

- [121] R. Myers, D. Montgomery, Response surface methodology, John Wiley & Sons, 1985.
- [122] A. Ratle, Kriging as a surrogate fitness landscape in evolutionary optimization, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol. 15, pp. 37–49, 2001.
- [123] C. Bishop, Neural networks for pattern recognition, Oxford press, 1995.
- [124] W. Carpenter, J.F. Barthelemy, A comparison of polynomial approximation and artificial neural nets as response surface AIAA Technical Report, vol. pp. 92–2247, 1992.
- [125] T. Simpson, T. Mauery, J. Korte, F. Mistree, Comparison of response surface and Kriging models for multidisciplinary design optimization AIAA Technical Report, pp. 98–4755, 1998.
- [126] D. Lim, Y. Jin, Y.-S. Ong, B. Sendhoff, Generalizing surrogate assisted evolutionary computation, IEEE Trans. Evol. Comput., vol. 14, no. 3, pp. 329–355, Jun. 2010.
- [127] J. Knowles, H. Nakayama, Meta-modeling in multiobjective optimization,” in Multiobjective Optimization (LNCS 5252). Berlin, Germany: Springer, pp. 245–284, 2008.
- [128] S.M. Elsayed, T. Ray, R.A. Sarker, Surrogate-assisted Differential Evolution Algorithm with Dynamic Parameters Selection for Solving Expensive Optimization Problems, in Proc. IEEE Congr. Evol. Comput., 2014.
- [129] H.K. Singh, A. Isaacs, T. Ray, A Hybrid Surrogate Based Algorithm (HSBA) to Solve Computationally Expensive Optimization Problems, in Proc. IEEE Congr. Evol. Comput., 2014.

- [130] W. Gong, A. Zhou, Z. Cai, A Multioperator Search Strategy Based on Cheap Surrogate Models for Evolutionary Optimization,"IEEE Trans. Evol. Comput., vol. 19, no. 5, pp. 746–758, 2015.
- [131] R. Mallipeddi, M. Lee, An evolving surrogate model-based differential evolution algorithm, Appl. Soft Comput., vol. 34, pp. 770–787, 2015.
- [132] X.Y. Nesterov, A. Nemirovskii, Y. Ye, Interior-point polynomial algorithms in convex programming. SIAM, vol. 13, 1994.
- [133] R. Mallipeddi, P.N. Suganthan, Ensemble Differential Evolution Algorithm for CEC2011 Problems, in Proc. IEEE Congr. Evol. Comput., 2011.
- [134] U. Halder, S. Das, D. Maity, A. Abraham, P. Dasgupta, Self Adaptive Cluster Based and Weed Inspired Differential Evolution Algorithm For Real World optimization, in Proc. IEEE Congr. Evol. Comput., 2011.
- [135] A. LaTorre, S. Muelas, J.-M. Pena, Benchmarking a Hybrid DE-RHC Algorithm on Real World Problems, in Proc. IEEE Congr. Evol. Comput., 2011.
- [136] K. Deb, Optimal design of a welded beam via genetic algorithms, Am. Inst. Aeronaut. Astronaut., vol. 11, pp. 2013–2015, 1991.
- [137] E. Sandgren, Nonlinear integer and discrete programming in mechanical design optimization, J. Mech. Des., vol. 112, 1990.
- [138] A. H. Gandomi, X. S. Yang and A. Alavi, Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems, Engineering with Computers, vol. 29, pp. 17–35, 2003.

- [139] G. Tomassetti, A cost-effective algorithm for the solution of engineering problems with particle swarm optimization, *Eng. Optimiz.* vol. 42, pp. 471–495, 2010.
- [140] A. Kaveh, S. Talatahari, Hybrid charged system search and particle swarm optimization for engineering design problems, *Int. J. Comput. Aided Eng. Softw.* vol. 28, pp. 423–440, 2011.
- [141] V.S. Aragon, S.C. Esquivel, C.A.C. Coello, A modified version of a T-Cell algorithm for constrained optimization problems, *Int. J. Numer. Meth. Eng.*, vol. 84, pp. 351–378, 2010.
- [142] C.A.C. Coello, E.M. Montes, Constraint-handling in genetic algorithms through the use of dominance-based tournament selection, *Advan. Eng. Inform.* vol. 16, pp. 193–203, 2002.
- [143] F.-z. Huang, L. Wang, Q. He, An effective co-evolutionary differential evolution for constrained optimization, *Appl. Math. Comput.*, vol. 186, pp. 340–356, 2007.
- [144] R.A. Krohling, L. dos Santos Coelho, Coevolutionary particle swarm optimization using Gaussian distribution for solving constrained optimization problems, *IEEE Trans. Syst. Man Cybern. Part B: Cybern.*, vol. 36, pp. 1407–1416, 2006.

APPENDIX A: Pseudocodes

Pseudocode A. 1: Pseudocode of canonical DE algorithm

Algorithm: Canonical DE	
1.	Step 1: Initialization
2.	Set population size, NP
3.	Initialize population at first generation g_0 , $POP_{g_0} = \langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ as shown in Eq.(1.3)
4.	Evaluate the initial population by computing the function values for each individual: $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$
5.	Sort POP_{g_0} based on computed function values, and mark the best individual as X_{best,g_0}
6.	While termination criterion is not met Do
7.	$g = g+1$
8.	For $i=1$ to NP
9.	Step 2: Mutation operation
10.	Select the mutation strategy from Eq. (1.4) to Eq. (1.8)
11.	Set the value of scaling factor, $F_{i,g}$
12.	Generate new mutant vector $V_{i,g}$
13.	Step 3: Crossover operation
14.	Select the crossover type, <i>bin</i> (Eq. (1.9)) or <i>exp</i> (Eq. (1.10))
15.	Set the value of crossover rate $CR_{i,g}$
16.	Generate the offspring $U_{i,g}$
17.	Step 5: Selection operation
18.	Evaluate the offspring $U_{i,g}$
19.	If $f(U_{i,g}) \leq f(X_{i,g})$
20.	Replace the target vector by newly generated offspring, $X_{i,g+1} = U_{i,g}$
21.	Mark the best individual:
22.	If $f(U_{i,g}) \leq f(X_{best,g})$
23.	$X_{best,g} = U_{i,g}$
24.	EndIf
25.	Else
26.	Keep the current target vector, $X_{i,g+1} = X_{i,g}$
27.	EndIf
28.	EndFor
29.	EndWhile
30.	Terminate and output $X_{best,g}$ and $f(X_{best,g})$

Pseudocode A. 2: Pseudocode of SFS-DP_{DE-GW} algorithm

Algorithm: SFS-DP_{DE-GW} Algorithm	
1.	Initialization
2.	Set population size, NP and maximum diffusion γ
3.	Initialize population at first generation g_0 , $POP_{g_0} = \langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ as shown in Eq. (1.3)
4.	Evaluate the initial population by computing the function values for each individual: $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$
5.	Sort POP_{g_0} based on computed function values, and mark the best individual as X_{best,g_0}
6.	Initialize all values in memory: μF and μCR to 0.5
7.	While termination criterion is not met Do
8.	Call differential evolution diffusion process
9.	For $k = 1$ to γ
10.	If $rand < 0.5$
11.	For $i=1$ to NP
12.	Call the Gaussian walks to generate new individual
13.	If $w < 0.5$
14.	Generate $X_{i,new}$ based on Eq. (2.1)
15.	Else
16.	Generate $X_{i,new}$ based on Eq. (2.2)
17.	EndIf
18.	Evaluate $X_{i,new}$ by computing its function value, $f(X_{i,new})$
19.	If $f(X_{i,new}) \leq f(X_i)$
20.	$X_{i,g+1} = X_{i,new}$
21.	Else
22.	$X_{i,g+1} = X_{i,g}$
23.	EndIf
24.	EndFor
25.	ElseIf
26.	Call DE operates (SHADE mutation)
27.	Reset $S_F = \phi$ and $S_{CR} = \phi$
28.	For $i = 1$ to NP
29.	Generate a random index $r_i = rand(1, H)$
30.	Generate $F_{i,g}$ and $CR_{i,g}$ using Cauchy and normal distributions as shown in Eq. (1.14) and (1.15) respectively
31.	Apply current-to-pbest with bin crossover, Eq. (1.12), to generate new offspring $U_{i,g}$
32.	Evaluate $U_{i,g}$ by computing its function value, $f(U_{i,g})$
33.	If $f(U_{i,g}) \leq f(X_i)$
34.	$X_{i,g+1} = U_{i,g}$
35.	Store successful control settings, $F_i \rightarrow S_F$ and $CR_i \rightarrow S_{CR}$
31.	Mark the best individual:
32.	If $f(U_{i,g}) \leq f(X_{best,g})$
33.	$X_{best,g} = U_{i,g}$
34.	EndIf
36.	ElseIf
37.	$X_{i,g+1} = X_{i,g}$

```

38.   EndIf
39.   EndFor
40.   Update memory of  $\mu F$  and  $\mu CR$  as shown in Eq. (1.20) - (1.25)
41.   EndIf
42.   EndFor
43.   Call update process of SFS
44.   Call first update process
45.   For  $i = 1$  to  $NP$ 
46.       Rank  $X_{i,g}$  and Compute  $Pa_i$  based on Eq. (2.4)
47.       For  $j = 1$  to  $D$ 
48.           If  $rand < Pa_i$ 
49.               Update index  $X_{i,g}^j$  as shown in Eq. (2.5)
50.           EndIf
51.       EndFor
52.       Evaluate  $X_{i,new,g}$  by calling its function value,  $f(X_{i,new,g})$ 
53.       If  $f(X_{i,new,g}) \leq f(X_i)$ 
54.            $X_{i,g} = X_{i,new,g}$ 
55.       EndIf
56.   EndFor
57.   Call second update process
58.   For  $i = 1$  to  $NP$ 
59.       Rank  $X_{i,g}$  and Compute  $Pa_i$  based on Eq. (2.4)
60.       If  $rand < Pa_i$ 
61.           Update  $X_{i,g}$  as shown in Eq. (2.6)
62.       If  $f(X_{i,new,g}) \leq f(X_i)$ 
63.            $X_{i,g} = X_{i,new,g}$ 
64.       EndIf
65.   EndIf
66.   EndFor
67. EndWhile
68. Terminate and output  $X_{best,g}$  and  $f(X_{best,g})$ 

```

Pseudocode A. 3: Pseudocode of *d*DSF-EA algorithm

Algorithm: <i>d</i>DSF-EA	
1. Initialization	
2. Set population size parameters, $NP_{\max}$, $NP_{\min}$ and maximum diffusion γ	
3. Set population size at first generation, $NP_{g_0} = NP_{\max}$	
4. Initialize population at first generation g_0 , $POP_{g_0} = \langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ as shown in Eq. (1.3)	
5. Evaluate the initial population by computing the function values for each individual: $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$	
6. Sort POP_{g_0} based on computed function values, and mark the best individual as X_{best,g_0}	
7. Initialize all values in memory: μF and μCR to 0.5	
8. While termination criterion is not met	
9. Call differential evolution diffusion process as shown in Lines 7-41 of Pseudocode A. 2	
10. Call success-based update process	
11. If $g = g_0$	
12. Set participation ratio for first and second update processes, $\Phi_{1,g_0} = \Phi_{2,g_0} = NP / 2$	
13. Call first update process as presented in Lines 43-55 of Pseudocode A. 2	
14. Call second update process as presented in Lines 56-65 of Pseudocode A. 2	
15. Else	
16. Compute participation ratio for first and second update processes, $\Phi_{1,g}$, $\Phi_{2,g}$ as shown in Eq. (2.7)	
17. Call first update process as presented in Lines 43-55 of Pseudocode A. 2	
18. Call second update process as presented in Lines 56-65 of Pseudocode A. 2	
19. EndIf	
20. Call population size dynamic reduction	
21. Calculate population size for next generation, NP_{g+1} as shown in Eq. (1.27)	
22. Sort individuals in POP_g based on function values	
23. Eliminate $(NP_g - NP_{g+1})$ worst individuals from POP_g	
24. EndWhile	
25. Terminate and output $X_{best,g}$ and $f(X_{best,g})$	

Pseudocode A. 4: Pseudo-code of LSHADE-EpSin algorithm

Algorithm: LSHADE-EpSin
<ol style="list-style-type: none"> 1. Initialization 2. Set population size parameters, $NP_{\max}, NP_{\min}$ 3. Set population size at first generation, $NP_{g_0} = NP_{\max}$ 4. Initialize population at first generation g_0, $POP_{g_0} = \langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ as shown in Eq. (1.3) 5. Evaluate the initial population by computing the function values for each individual: $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$ 6. Sort POP_{g_0} based on computed function values, and mark the best individual as X_{best,g_0} 7. Initialize all values in memory: μF and μCR to 0.5 8. Initialize mean value for sinusoidal adaptation: $freq$ and $\mu freq_{i,g_0}$ 0.5 9. Set N to 20 (the time to activate local search) 10. While termination criterion is not met Do 11. If $g \leq (g_{\max} / 2)$ 12. If $rand \leq 0.5$ 13. Generate $F_{i,g}$ using non-adaptive sinusoidal decreasing adjustment as shown in Eq. (2.8) 14. Else 15. Generate $freq_{i,g}$ using Eq. (2.10) 16. Generate $F_{i,g}$ using adaptive sinusoidal increasing adjustment as shown in Eq. (2.9) 17. EndIf 18. Else 19. Reset successful mean vectors, $S_F, S_{CR} = \phi$ 20. Generate a random index $r_i = rand(1, H)$ 21. Generate $F_{i,g}$ using Cauchy distribution as shown in Eq. (1.14) 22. EndIf 23. Generate $CR_{i,g}$ using Normal distribution as shown in Eq. (1.15) 24. For $i=1$ to NP 25. Apply current-to-pbest with bin crossover, Eq. (1.12), to generate new offspring $U_{i,g}$ 26. Evaluate $U_{i,g}$ by computing its function value, $f(U_{i,g})$ 27. If $f(U_{i,g}) \leq f(X_i)$ 28. $X_{i,g+1} = U_{i,g}$ 29. Store successful control settings, $F_i \rightarrow S_F$, $CR_i \rightarrow S_{CR}$ and $freq_{i,g} \rightarrow S_{freq}$ 30. Mark the best individual: 31. If $f(U_{i,g}) \leq f(X_{best,g})$ 32. $X_{best,g} = U_{i,g}$ 33. EndIf 34. ElseIf 35. $X_{i,g+1} = X_{i,g}$ 36. EndIf 37. EndFor 38. Update memory of μF and μCR as shown in Eq. (1.20) - (1.25) 39. Update $\mu freq$ as shown in Eq. (2.10) 40. Call population size dynamic reduction

41. **Calculate** population size for next generation, NP_{g+1} as shown in Eq. (1.27)
42. Sort individuals in POP_g based on function values
43. Eliminate $(NP_g - NP_{g+1})$ worst individuals from POP_g
44. **Call** local search
45. **If** $NP \leq N$ for the first time
46. Initialize 10 random individuals using Eq. (1.3) denoted as $X_i, i \in [1,10]$
47. Perform Gaussian walk for G_{LS} generations as shown in Eq. (2.1)
48. Evaluate new generated individuals
49. Pick the 10 worst individuals from original population
50. If newly generated individuals have better function value, replacement occurs
51. **EndIf**
52. **EndWhile**
53. **Terminate and output** $X_{best,g}$ and $f(X_{best,g})$

Pseudocode A. 5: Pseudo-code of EsDER-NR algorithm

Algorithm: EsDER-NR	
1. Initialization	
2. Set population size parameters, $NP_{\max}$, $NP_{\min}$	
3. Set population size at first generation, $NP_{g_0} = NP_{\max}$	
4. Initialize population at first generation g_0 , $POP_{g_0} = \langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ as shown in Eq. (1.3)	
5. Evaluate the initial population by computing the function values for each individual: $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$	
6. Sort POP_{g_0} based on computed function values, and mark the best individual as X_{best,g_0}	
7. Initialize all values in memory: μF and μCR to 0.5	
8. Initialize mean value for sinusoidal adaptation: $freq$ and $\mu freq_{i,g_0}$ 0.5	
9. Set N to 20 (time to activate local search)	
10. While termination criterion is not met Do	
11. Call evolutionary search with ensemble sinusoidal parameter adaptation as introduced in LSHADE-EpSin pseudocode (Lines 10-38)	
12. Call niching-based population size reduction	
13. If $g > \frac{g_{\max}}{2}$	
14. Pick the best individual from POP_g , $X_{best,g}$	
15. Compute the Euclidean distance, ED_i , for each $X_{i,g}$ in POP_g using Eq. (2.11)	
16. Sort POP_g in an ascending order based on computed ED_i	
17. Fill the first niche with the $X_{best,g}$ and its neighbours based on ED_i with size $NP/2$	
18. Fill the second niche with the remaining individuals	
19. Apply Eq. (1.27) to calculate population size for next generation, NP_{g+1}	
20. Calculate number of eliminated individuals, $NP_g - NP_{g+1}$	
21. Pick the best individual from the second niche	
22. Compute the Euclidean distance for each individual in the second niche	
23. Mark the neighbourhood region of best individual as the elimination region	
24. Eliminate $NP_g - NP_{g+1}$ individuals from the marked elimination region of second niche	
25. EndIf	
26. If $NP \leq N$ for the first time	
54. Call local search based on Gaussian walks as presented pseudocode of LSHADE-EpSin (Lines 45-49)	
27. EndIf	
28. EndWhile	
29. Terminate and output $X_{best,g}$ and $f(X_{best,g})$	

Pseudocode A. 6: Pseudocode of L-covnSHADE algorithm

Algorithm: L-covnSHADE

1. **Initialization**
2. Set population size, NP and maximum diffusion γ
3. Initialize population at first generation g_0 , $POP_{g_0} = \langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ as shown in Eq. (1.3)
4. Evaluate the initial population by computing the function values for each individual:
 $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$
5. Sort POP_{g_0} based on computed function values, and mark the best individual as X_{best,g_0}
6. Initialize all values in memory: μF and μCR to 0.5
7. Initialize covariance matrix settings, $ps = 0.5$, $pa = 0.4$
8. **While** termination criterion is not met **Do**
9. **Reset** $S_F = \phi$ and $S_{CR} = \phi$
10. **For** $i = 1$ to NP
11. Generate a random index $r_i = rand(1, H)$
12. Generate $F_{i,g}$ and $CR_{i,g}$ using Cauchy and normal distributions as shown in Eq. (1.14) and (1.15) respectively
13. Apply current-to- p best, Eq. (1.12), to generate new mutant vector $V_{i,g}$
14. **Call covariance matrix learning for crossover operator**
15. **If** $rand < pa$
16. Mark best individual $X_{best,g}$
17. Compute the Euclidean distance, ED_i , for each $X_{i,g}$ in POP_g using Eq. (2.11)
18. Select $NP \times ps$ to form covariance matrix as shown in Eq. (2.12)
19. Transform $X_{i,g}$ and $V_{i,g}$ as shown in Eq. (2.13) and (2.14) to $\bar{X}_{i,g}$ and $\bar{V}_{i,g}$
20. Apply bin crossover on $\bar{X}_{i,g}$ and $\bar{V}_{i,g}$ as shown in Eq. (2.15)
21. Transform $\bar{U}_{i,g}$ to original coordinate system, $U_{i,g}$
22. **Else**
23. Apply binomial crossover on original coordinate system, $X_{i,g}$ and $V_{i,g}$ as shown in Eq. (1.9)
24. **EndIf**
25. Evaluate $U_{i,g}$ by computing its function value, $f(U_{i,g})$
26. **If** $f(U_{i,g}) \leq f(X_i)$
27. $X_{i,g+1} = U_{i,g}$
28. Store successful control settings, $F_i \rightarrow S_F$ and $CR_i \rightarrow S_{CR}$
29. Mark the best individual:
30. **If** $f(U_{i,g}) \leq f(X_{best,g})$
31. $X_{best,g} = U_{i,g}$
32. **EndIf**
33. **ElseIf**
34. $X_{i,g+1} = X_{i,g}$
35. **EndIf**
36. **EndFor**
37. Update memory of μF and μCR as shown in Eq. (1.20) - (1.25)
38. **Call population size dynamic reduction**
39. **Calculate** population size for next generation, NP_{g+1} as shown in Eq. (1.27)
40. Sort individuals in POP_g based on function values

<p>41. Eliminate $(NP_g - NP_{g+1})$ worst individuals from POP_g</p> <p>42. EndWhile</p> <p>43. Terminate and output $X_{best,g}$ and $f(X_{best,g})$</p>
--

Pseudocode A. 7: Pseudocode of iDEaSm algorithm

Algorithm: iDEaSm	
1. Initialization	
2. Set population size parameters, $NP_{\max}, NP_{\min}$	
3. Set population size at first generation, $NP_{g_0} = NP_{\max}$	
4. Initialize population at first generation g_0 , $POP_{g_0} = \langle X_{1,g_0}, X_{2,g_0}, \dots, X_{NP,g_0} \rangle$ as shown in Eq. (1.3)	
5. Evaluate the initial population by computing the function values for each individual: $\langle f(X_{1,g_0}), f(X_{2,g_0}), \dots, f(X_{NP,g_0}) \rangle$	
6. Sort POP_{g_0} based on computed function values, and mark the best individual as X_{best,g_0}	
7. Initialize all values in memory: μF and μCR to 0.5	
8. While termination criterion is not met Do	
9. Build the adapted surrogate model	
10. Use $\{X_1, X_2, \dots, X_{NP}\}, \{f(X_1), f(X_2), f(X_{NP})\}$ as an input to Kriging model	
11. Use steps 1-4 as explained in sub-section 10.3.1 to adapt the θ parameter	
12. Compute the function of interest $y(X)$ as shown in Eqs. (3.4) - (3.5)	
13. For $i=1$ to NP	
14. $q = 1$	
15. While $q > w$ or $flag == true$	
16. Generate $F_{i,g}$ and $CR_{i,g}$ using Cauchy and normal distributions as shown in (1.14) and (1.15) respectively	
17. Apply current-to-pbest, Eq. (1.12), to generate new mutant vector $V_{i,g}$	
18. Apply binomial crossover to generate offspring $U_{i,g}$ as shown in Eq. (1.9)	
19. Evaluate the new offspring using the surrogate model, $y(U_{i,g})$	
20. If $y(U_{i,g}) \leq f(X_{i,g})$	
21. $flag = true$	
22. EndIf	
23. $q = q + 1$	
24. EndWhile	
25. Evaluate $U_{i,g}$ using the original objective function, $f(U_{i,g})$	
26. If $f(U_{i,g}) \leq f(X_{i,g})$	
27. $X_{i,g} = U_{i,g}$	
28. Save the successful values for $F_{i,g}$ and $CR_{i,g}$ in S_F and S_{CR} respectively	
29. EndIf	
30. EndFor	
31. Update memory of μF and μCR as shown in Eq. (1.20) - (1.25)	
32. Perform local search	
33. Pick the best solution, $X_{best,g}$	
34. Perform the interior point method to generate new solution, X_{new} and evaluate it using the surrogate model, $y(X_{new})$	
35. Evaluate the final solution using the original function value $f(X_{new})$	

36. **If** $f(X_{new}) \leq f(X_{best,g})$
37. $X_{best,g} = X_{new}$
38. **EndIf**
39. **Call population size dynamic reduction**
40. **Calculate** population size for next generation, NP_{g+1} as shown in [Eq. \(1.27\)](#)
41. Sort individuals in POP_g based on function values
42. Eliminate $(NP_g - NP_{g+1})$ worst individuals from POP_g
43. **EndWhile**
44. **Terminate and output** $X_{best,g}$ and $f(X_{best,g})$

APPENDIX B: Benchmark Functions

Table B. 1: Summary of CEC'14 Test Functions [13]

	No.	Functions	$F_i^*=F_i(x^*)$
Unimodal Functions	1	Rotated High Conditioned Elliptic Function	100
	2	Rotated Bent Cigar Function	200
	3	Rotated Discus Function	300
Simple Multimodal Functions	4	Shifted and Rotated Rosenbrock's Function	400
	5	Shifted and Rotated Ackley's Function	500
	6	Shifted and Rotated Weierstrass Function	600
	7	Shifted and Rotated Griewank's Function	700
	8	Shifted Rastrigin's Function	800
	9	Shifted and Rotated Rastrigin's Function	900
	10	Shifted Schwefel's Function	1000
	11	Shifted and Rotated Schwefel's Function	1100
	12	Shifted and Rotated Katsuura Function	1200
	13	Shifted and Rotated HappyCat Function	1300
	14	Shifted and Rotated HGBat Function	1400
	15	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function	1500
	16	Shifted and Rotated Expanded Scaffer's F6 Function	1600
Hybrid Function 1	17	Hybrid Function 1 ($N=3$)	1700
	18	Hybrid Function 2 ($N=3$)	1800
	19	Hybrid Function 3 ($N=4$)	1900
	20	Hybrid Function 4 ($N=4$)	2000
	21	Hybrid Function 5 ($N=5$)	2100
	22	Hybrid Function 6 ($N=5$)	2200
Composition Functions	23	Composition Function 1 ($N=5$)	2300
	24	Composition Function 2 ($N=3$)	2400
	25	Composition Function 3 ($N=3$)	2500
	26	Composition Function 4 ($N=5$)	2600
	27	Composition Function 5 ($N=5$)	2700
	28	Composition Function 6 ($N=5$)	2800
	29	Composition Function 7 ($N=3$)	2900
	30	Composition Function 8 ($N=3$)	3000
Search Range: $[-100,100]^D$			

Table B. 2: Summary of CEC'11 real-world numerical optimization problems

[112]

Problem No.	Description	<i>D</i>
T01	Parameter Estimation of FM Sound Waves	6
T02	Lennard-Jones Potential	30
T03	Bi-functional Catalyst Blend Optimal Control	1
T04	Optimal Control of a Non-Linear Stirred Tank Reactor	1
T05	Tersoff Potential Function Min. Problem-Si(B)	30
T06	Tersoff Potential Function Min. Problem-Si(C)	30
T07	Spread Spectrum Radar Polly phase Code Design	20
T08	Transmission Network Expansion Planning	7
T09	Large Scale Transmission Pricing	126
T10	Circular Antenna Array Design	12
T10.1	Dynamic Economic Dispatch - Instance 1	120
T10.2	Dynamic Economic Dispatch - Instance 2	240
T11.1	Static Economic Load Dispatch – Instance 1	6
T11.2	Static Economic Load Dispatch – Instance 2	13
T11.3	Static Economic Load Dispatch – Instance 3	15
T11.4	Static Economic Load Dispatch – Instance 4	40
T11.5	Static Economic Load Dispatch – Instance 5	140
T12.1	Hydrothermal Scheduling - Instance 1	96
T12.2	Hydrothermal Scheduling - Instance 2	96
T12.3	Hydrothermal Scheduling - Instance 3	96
T13	Messenger: Spacecraft Trajectory Optimization	26
T14	Cassini 2: Spacecraft Trajectory Optimization	22

Table B. 3: Summary for functions description for bound-constrained
benchmark set [11]

	Function		$f(X^*)$	Search Range
f_1	Sphere Function	$f_1(x) = \sum_{i=1}^D x_i^2$	0	$[-100,100]^D$
f_2	Generalized Rosenbrock Function	$f_2(x) = \sum_{i=1}^{D-1} (100(x_i^2 - x_{i+1}^2)^2 + (x_i - 1)^2)$	0	$[-100,100]^D$
f_3	Ackley function	$f_3(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2}\right) - \exp\left(\frac{1}{D} \sqrt{\sum_{i=1}^D \cos(2\pi x_i)}\right) + 20 + e$	0	$[-32,32]^D$
f_4	Generalized Griewank Function	$f_4(x) = \sum_{i=1}^D \frac{x_i^2}{4000} - \prod_{i=1}^D \frac{x_i}{\sqrt{i}} + 1$	0	$[-600,600]^D$
f_5	Generalized Rastrigin Function	$f_5(x) = \sum_{i=1}^D (x_i^2 - 10 \cos(2\pi x_i) + 10)$	0	$[-5.12, 5.12]^D$
f_6	Non-continuous Rastrigin Function	$f_6(x) = \sum_{i=1}^D (y_i^2 - 10 \cos(2\pi y_i) + 10)$ $y_i = \begin{cases} x_i & x_i < 0.5 \\ \frac{\text{round}(2x_i)}{2} & x_i \geq 0.5 \end{cases}$	0	$[-500,500]^D$
f_7	Schwefel's Function 2.26	$f_7(x) = 418.9829 \times D - \sum_{i=1}^D x_i \sin(x_i ^{0.5})$	0	$[-500,500]^D$
f_8	Schwefel's Problem 1.2	$f_8(x) = \sum_{i=1}^D \left(\sum_{j=1}^i x_j\right)^2$	0	$[-100,100]^D$
f_9	Schwefel's Problem 1.2 with Noise in Fitness	$f_9(x) = \left(\sum_{i=1}^D \left(\sum_{j=1}^i x_j\right)^2\right) (1 + 0.4 N(0,1))$	0	$[-100,100]^D$
f_{10}	High Conditioned Elliptic Function	$f_{10}(x) = \sum_{i=1}^D (10^6)^{\frac{(i-1)(D-1)}{2}} x_i^2$	0	$[-100,100]^D$
f_{11}	Weierstrass's Function	$f_{11}(x) = \sum_{i=1}^D \left(\sum_{k=0}^{k_{\max}} \left[a^k \cos(2\pi b^k (x_i + 0.5))\right]\right) - D \sum_{k=0}^{k_{\max}} \left[a^k \cos(2\pi b^k \cdot 0.5)\right]$	0	$[-0.5,0.5]^D$
f_{12}	Schwefel's Problem 2.22	$f_{12}(x) = \sum_{i=1}^D x_i + \prod_{i=1}^D x_i $	0	$[-10,10]^D$
f_{13}	Schwefel's Problem 2.21	$f_{13}(x) = \max_i \{ x_i , 1 \leq i \leq D\}$	0	$[-100,100]^D$

f_{14}	Generalized Penalized Function 1	$f_{14}(x) = \frac{\pi}{D} \left\{ 10 \sin^2(\pi y_1) + \sum_{i=1}^{D-1} (y_i - 1)^2 [1 + 10(\pi y_{i+1})] \right.$ $\left. + (y_D - 1)^2 + \sum_{i=1}^D u(x_i, 10, 100, 4) \right.$ $y_i = 1 + 0.25(x_i + 1)$ $u(x_i, 10, 100, 4) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \leq x_i \leq a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	0	$[-50, 50]^D$
f_{15}	Generalized Penalized Function 2	$f_{15}(x) = 0.1 \left\{ \sin^2(3\pi x_1) + \sum_{i=1}^{D-1} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] \right.$ $\left. + (x_D - 1)^2 [1 + \sin^2(2\pi x_D)] + \sum_{i=1}^D u(x_i, 5, 100, 4) \right.$	0	$[-50, 50]^D$

APPENDIX C: Algorithms Complexity

The computational complexity of an algorithm is calculated as described in [13] for all the dimensions ($10D$, $30D$, $50D$ and $100D$) as shown in any table of this appendix. T_0 denotes the running time of the following program:

```
for i = 1:1000000
    x = 0.55 + (double)i; x = x + x; x = x / 2; x = x * x;
    x = sqrt(x); x = log(x); x = exp(x); x = x / (x + 2);
end
```

T_1 is the computing time for f_{18} for 200,000 evaluations while T_2 is the complete running time for the proposed algorithm of f_{18} for 200,000 evaluations. T_2 is evaluated five times, and the mean for T_2 is denoted as $\hat{T}_2$. Finally, the algorithm complexity is reflected by $\hat{T}_2, T_1$ and $(\hat{T}_2 - T_1)/T_0$.

Table C. 1: Algorithm complexity of SFS-DP_{DE-GW}

	T_0	T_1	$\hat{T}_2$	$(\hat{T}_2 - T_1)/T_0$
$D = 10$	0.1601	0.6234	5.0236	27.48407
$D = 30$		0.8123	5.6346	30.12055
$D = 50$		1.2463	6.8463	34.97814
$D = 100$		2.6472	11.2416	53.68145

Table C. 2: Algorithm complexity of dDSF-EA

	T_0	T_1	$\hat{T}_2$	$(\hat{T}_2 - T_1)/T_0$
$D = 10$	0.1548	0.6153	4.5781	25.5995
$D = 30$		0.7995	5.0081	27.1873
$D = 50$		1.1176	5.9727	31.3637
$D = 100$		2.7216	10.3792	49.4677

Table C. 3: Algorithm complexity of LSHADE-EpSin

	T_0	T_1	$\hat{T}_2$	$(\hat{T}_2 - T_1)/T_0$
$D = 10$	0.1598	0.6239	1.0867	2.8963
$D = 30$		1.0129	2.6938	10.5186
$D = 50$		1.2899	3.0572	11.0596
$D = 100$		3.5622	7.2315	22.9616

Table C. 4: Algorithm complexity of EsDEr-NR

	T_0	T_1	$\hat{T}_2$	$(\hat{T}_2 - T_1)/T_0$
$D = 10$	0.1442	0.7142	1.4162	4.8682
$D = 30$		1.1253	2.7425	11.2150
$D = 50$		1.2456	3.1458	13.1775
$D = 100$		3.7125	6.9475	22.4341

APPENDIX D: Experimental Results Tables

A two-sided Wilcoxon rank sum test [87] is used to evaluate the statistical significance of the observed differences between the contestant algorithms. These tests show the result of the two-sided Wilcoxon rank sum test between our proposed algorithm (which is presented in the last column) and every other algorithm used in the comparisons. At the 0.05 significance level, a “–” sign signifies the case where the compared algorithm shows an inferior performance, and a “+” sign identifies when it exhibits superior performance. In such cases H_0 is rejected. When the performance difference is not statistically significant, these cases are marked with a “=” sign. Moreover, according to the obtained Wilcoxon’s test, the results are summarized as “ $w/t/l$ ” in the last row of each of the tables, to imply that the contestant algorithm wins in w functions, ties in t functions, and loses in l functions, in a pairwise comparison with our proposed algorithm in the last column.

Table D. 1: A comparison of SFS-DP_{DE-GW} and other algorithms on CEC'14

benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 30D
averaged over 51 runs. Best entries are marked in bold.

	SaDE	CoDE	EPSDE	JADE	SHADE	SFS	SFS-DP _{DE-GW}
f_1	2.9897E+05 (2.3297E+05) -	2.6332E+04 (1.8386E+04) -	2.4162E+04 (8.2591E+04) -	4.4782E+02 (1.0397E+03) -	4.8134E+02 (7.8643E+02) -	2.1491E+03 (1.0207E+03) -	0.0000E+00 (0.0000E+00) -
f_2	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	5.1611E-02 (2.8378E-02) -	0.0000E+00 (0.0000E+00) -
f_3	1.4258E+01 (5.8311E+01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	5.6292E-04 (2.5067E-03) -	0.0000E+00 (0.0000E+00) =	3.408E-03 (1.791E-03) -	0.0000E+00 (0.0000E+00) -
f_4	3.7184E+01 (3.7647E+01) -	2.5174E+00 (1.2583E+01) -	3.2127E+00 (2.2348E+00) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	3.485E-01 (4.461E-01) -	0.0000E+00 (0.0000E+00) -
f_5	2.0536E+01 (5.5479E-02) =	2.0064E+01 (9.7834E-02) =	2.0347E+01 (3.2166E-02) =	2.0287E+01 (3.7919E-02) =	2.0101E+01 (1.9092E-02) =	2.051E+01 (5.064E-02) =	2.0121E+01 (1.8001E-02) -
f_6	5.4599E+00 (1.9996E+00) -	1.9891E+00 (1.6744E+00) -	1.8893E+01 (1.6361E+00) -	9.4229E+00 (2.1611E+00) -	5.2891E-01 (7.2172E-01) -	8.656E+00 (3.762E+00) -	0.0000E+00 (0.0000E+00) -
f_7	1.2333E-02 (1.9953E-02) -	1.4502E-04 (1.0357E-03) -	2.0760E-03 (5.5347E-03) -	0.0000E+00 (0.0000E+00) =	4.8332E-04 (1.9731E-03) -	1.450E-04 (1.036E-03) -	0.0000E+00 (0.0000E+00) -
f_8	7.8036E-02 (3.3553E-01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	8.281E-02 (6.178E-02) -	0.0000E+00 (0.0000E+00) -
f_9	3.8121E+01 (9.1222E+00) -	4.0371E+01 (1.2218E+01) -	4.4362E+01 (6.7553E+00) -	2.6166E+01 (4.1785E+00) -	1.5832E+01 (3.1702E+00) -	4.861E+01 (9.015E+00) -	1.3656E+01 (2.3594E+00) -
f_{10}	2.6919E-01 (4.6192E-01) -	5.0019E-01 (4.3629E-01) -	2.0140E-01 (2.2970E-01) -	5.3069E-03 (1.0065E-02) -	1.2658E-02 (1.6694E-02) -	1.392E+01 (3.829E+00) -	5.6532E-04 (3.1027E-03) -
f_{11}	3.1475E+03 (6.9708E+02) -	1.9513E+03 (4.8759E+02) -	3.5646E+03 (3.4162E+02) -	1.6399E+03 (2.6440E+02) -	1.3969E+03 (2.1231E+02) +	2.246E+03 (2.289E+02) -	1.4780E+03 (2.4940E+02) -
f_{12}	7.9419E-01 (1.1392E-01) -	5.9989E-02 (3.5730E-02) +	5.2516E-01 (7.9944E-02) -	2.7113E-01 (3.7518E-02) =	1.6227E-01 (2.4622E-02) =	5.575E-01 (7.923E-02) -	1.6376E-01 (2.3318E-02) -
f_{13}	2.5159E-01 (4.0437E-02) =	2.3130E-01 (5.0203E-02) =	2.4299E-01 (4.3511E-02) =	2.2030E-01 (3.9054E-02) =	2.0401E-01 (3.3979E-02) =	2.756E-01 (4.710E-02) =	1.9565E-01 (3.6867E-02) -
f_{14}	2.2853E-01 (3.6057E-02) =	2.3889E-01 (3.5058E-02) =	2.7812E-01 (6.1561E-02) =	2.3399E-01 (3.0026E-02) =	2.2468E-01 (2.8035E-02) =	2.103E-01 (2.742E-02) =	2.5100E-01 (3.3380E-02) -
f_{15}	4.1410E+00 (1.3877E+00) -	3.1752E+00 (8.2967E-01) =	5.6693E+00 (8.5685E-01) -	3.0980E+00 (4.2989E-01) =	2.5641E+00 (3.5823E-01) =	5.558E+00 (6.750E-01) -	2.4868E+00 (3.6992E-01) -
f_{16}	1.0911E+01 (2.7495E-01) =	9.2625E+00 (8.7692E-01) =	1.1146E+01 (3.0350E-01) -	9.3698E+00 (3.6416E-01) =	9.1478E+00 (4.6678E-01) =	1.048E+01 (2.710E-01) =	9.1553E+00 (4.4693E-01) -
f_{17}	1.1531E+04 (1.0350E+04) -	1.4530E+03 (1.6939E+03) -	4.6053E+04 (5.9842E+04) -	9.6734E+03 (5.9893E+04) -	1.0589E+03 (3.1817E+02) -	9.228E+02 (1.948E+02) -	3.6192E+02 (1.1614E+02) -
f_{18}	4.4383E+02 (6.5301E+02) -	1.3443E+01 (5.4219E+00) +	3.3188E+02 (3.9702E+02) -	3.5805E+02 (1.5087E+03) -	4.9875E+01 (2.4497E+01) -	2.821E+01 (6.398E+00) -	1.9726E+01 (3.9324E+00) -
f_{19}	4.0013E+00 (8.6299E-01) =	2.7041E+00 (4.5214E-01) +	1.3300E+01 (9.9535E-01) -	4.4373E+00 (6.7044E-01) =	4.3057E+00 (7.2316E-01) =	5.074E+00 (5.793E-01) =	4.7291E+00 (9.1087E-01) -
f_{20}	1.2454E+02 (9.5739E+01) -	1.0913E+01 (5.6106E+00) +	5.0042E+01 (8.1396E+01) -	2.8854E+03 (2.3250E+03) -	1.2642E+01 (6.8123E+00) +	2.641E+01 (3.946E+00) -	1.5477E+01 (2.2300E+00) -
f_{21}	2.8350E+03 -	2.0005E+02 -	8.8443E+03 -	7.5750E+03 -	2.5176E+02 -	4.301E+02 -	2.4576E+02 -

	(3.0596E+03) –	(1.7005E+02) +	(1.1976E+04) –	(3.7939E+04) –	(1.1511E+02) –	(1.029E+02) –	(1.2426E+02)
f_{22}	1.6633E+02 (5.4534E+01) –	1.9333E+02 (1.1256E+02) –	2.5561E+02 (8.3208E+01) –	1.4570E+02 (7.2532E+01) –	1.1419E+02 (5.5673E+01) –	1.241E+02 (6.656E+01) –	9.0100E+01 (6.4577E+01)
f_{23}	3.1524E+02 (5.0125E-13) =	3.1524E+02 (3.3677E-13) =	3.1401E+02 (8.9971E-13) =	3.1524E+02 (2.2964E-13) =	3.1524E+02 (2.2964E-13) =	3.152E+02 (1.425E-10) =	3.1524E+02 (4.0186E-13)
f_{24}	2.2862E+02 (5.0472E+00) –	2.2509E+02 (2.1852E+00) =	2.2884E+02 (6.6430E+00) –	2.2561E+02 (3.5926E+00) =	2.2494E+02 (1.9816E+00) =	2.237E+02 (6.597E-01) –	2.2389E+02 (1.1340E+00)
f_{25}	2.0828E+02 (2.3077E+00) –	2.0362E+02 (6.7389E-01) =	2.0023E+02 (4.0583E-02) =	2.0388E+02 (1.2730E+00) =	2.0342E+02 (7.3703E-01) =	2.033E+02 (2.358E-01) =	2.0316E+02 (4.7200E-02)
f_{26}	1.0614E+02 (2.3705E+01) –	1.0023E+02 (5.7694E-02) =	1.0026E+02 (3.9448E-02) =	1.0218E+02 (1.3974E+01) =	1.0020E+02 (3.3515E-02) =	1.003E+02 (4.940E-02) =	1.0019E+02 (2.0958E-02)
f_{27}	4.1786E+02 (3.7565E+01) –	3.8340E+02 (3.6817E+01) –	8.3334E+02 (1.1550E+02) –	3.4415E+02 (5.1560E+01) –	3.2454E+02 (3.8865E+01) –	4.002E+02 (7.977E-02) –	3.0196E+02 (1.4003E+01)
f_{28}	8.9045E+02 (2.9373E+01) –	8.3876E+02 (3.3236E+01) =	3.9831E+02 (1.4468E+01) +	8.0273E+02 (3.5775E+01) +	8.4047E+02 (3.1556E+01) –	8.918E+02 (3.056E+01) –	8.3445E+02 (1.7572E+01)
f_{29}	1.0792E+03 (1.8654E+02) –	7.8866E+02 (1.2518E+02) –	2.1430E+02 (1.1392E+00) +	7.5909E+02 (1.5931E+02) –	7.1516E+02 (6.4332E+01) =	3.780E+02 (6.259E+01) +	7.1637E+02 (2.0265E+00)
f_{30}	1.6680E+03 (5.7200E+02) –	9.9971E+02 (4.5398E+02) +	5.6708E+02 (1.1242E+02) +	1.9192E+03 (6.4143E+02) –	1.5656E+03 (7.2836E+02) –	8.598E+02 (1.534E+02) +	1.1006E+03 (4.2661E+02)
w/t/l	0/6/24	6/13/11	3/9/18	1/15/14	2/16/12	2/8/20	

Table D. 2: A comparison of SFS-DP_{DE-GW} and other algorithms on CEC'14

benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D
averaged over 51 runs. Best entries are marked in bold.

	SaDE	CoDE	EPSDE	JADE	SHADE	SFS	SFS-DV _{DE-GW}
f_1	9.3147E+05 (3.1419E+05) -	2.3214E+05 (1.0747E+05) -	1.7374E+06 (5.5064E+06) -	1.5275E+04 (1.3422E+04) -	1.8614E+04 (1.3638E+04) -	3.5520E+04 (2.9330E+04) -	2.7585E+00 (5.9779E+00)
f_2	3.7564E+03 (3.8115E+03) -	6.5010E+01 (1.9284E+02) -	1.4999E-08 (3.4609E-08) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	1.8218E+03 (1.3552E+03) -	0.0000E+00 (0.0000E+00)
f_3	3.4631E+03 (2.2424E+03) -	2.5387E+01 (4.1188E+01) -	3.7908E-04 (1.8292E-03) -	4.5134E+03 (2.3441E+03) -	0.0000E+00 (0.0000E+00) =	8.4882E-01 (4.2219E-01) -	0.0000E+00 (0.0000E+00)
f_4	8.2104E+01 (4.4460E+01) -	2.6067E+01 (3.3789E+01) -	4.1596E+01 (2.2985E+01) -	1.9346E+01 (3.9290E+01) =	9.7224E+00 (2.9435E+01) +	8.1011E+01 (2.4655E+01) -	1.5442E+01 (3.9417E+01)
f_5	2.0733E+01 (4.1549E-02) =	2.0032E+01 (6.8626E-02) =	2.0596E+01 (3.2400E-02) =	2.0356E+01 (3.6845E-02) =	2.0140E+01 (1.9410E-02) =	2.0639E+01 (5.5436E-02) =	2.0138E+01 (5.3443E-02)
f_6	1.8148E+01 (3.2634E+00) -	8.4983E+00 (3.3570E+00) -	4.5596E+01 (2.6277E+00) -	1.6589E+01 (6.6285E+00) -	5.1707E+00 (2.4284E+00) -	1.2087E+01 (5.0632E+00) -	1.9080E-01 (4.4651E-01)
f_7	1.3887E-02 (1.5352E-02) -	1.5467E-03 (3.1935E-03) -	5.1192E-03 (7.5766E-03) -	2.0769E-03 (4.9084E-03) -	3.9093E-03 (7.4077E-03) -	5.8008E-04 (2.0082E-03) -	0.0000E+00 (0.0000E+00)
f_8	1.0925E+00 (1.0945E+00) -	4.8773E-01 (8.2941E-01) -	1.9509E-02 (1.3932E-01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	8.9156E-02 (1.5473E-01) -	0.0000E+00 (0.0000E+00)
f_9	9.1673E+01 (1.4268E+01) -	8.2582E+01 (1.7230E+01) -	1.4593E+02 (1.6322E+01) -	5.1867E+01 (8.6673E+00) -	3.4182E+01 (6.1592E+00) =	1.0525E+02 (1.6659E+01) -	3.1522E+01 (4.4965E+00)
f_{10}	1.2284E+00 (1.0759E+00) -	5.1856E+00 (3.0648E+00) -	5.5334E+02 (6.0708E+02) -	1.2247E-02 (1.3099E-02) =	1.0287E-02 (1.1368E-02) =	2.1706E+01 (3.8438E+00) -	2.2883E-03 (4.8075E-03)
f_{11}	6.8299E+03 (1.5764E+03) -	4.3391E+03 (9.4713E+02) -	8.9455E+03 (5.3473E+02) -	3.8695E+03 (2.7785E+02) -	3.5552E+03 (3.2722E+02) -	4.7676E+03 (3.7755E+02) -	3.4391E+03 (3.0948E+02)
f_{12}	1.0986E+00 (1.0537E-01) -	8.7329E-02 (4.5691E-02) +	8.5469E-01 (6.9658E-02) -	2.5535E-01 (3.1321E-02) =	1.6366E-01 (2.1190E-02) =	6.1671E-01 (7.2547E-02) -	1.6510E-01 (2.0162E-02)
f_{13}	4.3681E-01 (5.4721E-02) -	3.3636E-01 (4.9736E-02) -	3.6268E-01 (5.9205E-02) -	3.3223E-01 (4.9590E-02) -	3.1034E-01 (4.5483E-02) -	3.8390E-01 (4.7927E-02) -	2.4304E-01 (3.0417E-02)
f_{14}	3.1170E-01 (3.2642E-02) =	2.7652E-01 (2.9839E-02) =	3.5209E-01 (9.0136E-02) =	3.1348E-01 (9.1988E-02) =	2.9453E-01 (5.7219E-02) =	2.5988E-01 (2.2820E-02) =	3.2397E-01 (8.3224E-02)
f_{15}	1.7092E+01 (5.8042E+00) -	7.4043E+00 (1.3955E+00) =	1.8616E+01 (2.6294E+00) -	7.0826E+00 (9.4236E-01) =	5.7376E+00 (6.1481E-01) =	1.2893E+01 (1.6214E+00) -	6.8975E+00 (1.3995E+00)
f_{16}	2.0222E+01 (2.7357E-01) -	1.8290E+01 (8.9113E-01) =	2.0695E+01 (4.1566E-01) -	1.7720E+01 (4.3341E-01) =	1.7444E+01 (4.2728E-01) =	1.9123E+01 (4.6960E-01) =	1.7425E+01 (4.7906E-01)
f_{17}	5.9118E+04 (3.9220E+04) -	1.5714E+04 (1.3291E+04) -	2.2167E+05 (1.4896E+05) -	2.3000E+03 (5.9502E+02) -	2.2347E+03 (7.8109E+02) -	2.0599E+03 (3.9736E+02) -	1.3574E+03 (3.9909E+02)
f_{18}	7.5884E+02 (5.1263E+02) -	3.2916E+02 (3.4839E+02) -	3.6879E+03 (8.1501E+03) -	1.8203E+02 (4.7150E+01) -	1.5375E+02 (3.6617E+01) -	7.2489E+01 (1.8364E+01) +	9.9619E+01 (1.5245E+01)
f_{19}	1.6741E+01 (8.7552E+00) -	6.1691E+00 (9.8380E-01) +	2.4584E+01 (1.6861E+00) -	1.3421E+01 (5.2199E+00) =	9.3810E+00 (3.2466E+00) =	1.0247E+01 (2.1614E+00) =	1.1124E+01 (3.0602E+00)
f_{20}	9.0611E+02 (7.3143E+02) -	2.1861E+02 (2.3515E+02) -	4.5080E+02 (5.6348E+02) -	7.9529E+03 (6.9640E+03) -	1.9816E+02 (5.7311E+01) -	7.4670E+01 (1.3192E+01) -	4.7599E+01 (8.9308E+00)
f_{21}	8.5745E+04	6.4912E+03	7.4370E+04	2.4322E+04	1.2297E+03	1.1860E+03	4.6959E+02

	(4.5410E+04) –	(5.5372E+03) –	(8.5331E+04) –	(1.6455E+05) –	(4.0604E+02) –	(2.0900E+02) –	(1.4816E+02)
f_{22}	5.1001E+02 (1.7759E+02) –	6.0223E+02 (2.2061E+02) –	8.0373E+02 (2.0458E+02) –	5.1982E+02 (1.3024E+02) –	3.6769E+02 (1.5946E+02) +	4.9241E+02 (1.4991E+02) –	4.0148E+02 (1.2036E+02)
f_{23}	3.4400E+02 (1.1482E-13) =	3.4400E+02 (1.1482E-13) =	3.3705E+02 (3.4523E-12) =	3.4400E+02 (3.7646E-13) =	3.4400E+02 (1.1482E-13) =	3.4400E+02 (1.6202E-12) =	3.4400E+02 (4.5927E-13)
f_{24}	2.7596E+02 (3.0905E+00) =	2.7139E+02 (2.9312E+00) =	2.7312E+02 (5.7937E+00) =	2.7482E+02 (1.9487E+00) =	2.7478E+02 (1.7334E+00) =	2.6156E+02 (4.6475E+00) +	2.7509E+02 (8.5349E-01)
f_{25}	2.1665E+02 (9.7720E+00) –	2.0792E+02 (4.7505E+00) =	2.0116E+02 (2.0229E+00) =	2.1699E+02 (6.7636E+00) –	2.1217E+02 (6.9165E+00) –	2.0746E+02 (6.7402E-01) =	2.0546E+02 (3.2251E-01)
f_{26}	1.9030E+02 (2.9941E+01) –	1.0618E+02 (2.3701E+01) =	1.0035E+02 (4.7521E-02) =	1.0237E+02 (1.3950E+01) =	1.0431E+02 (1.9529E+01) =	1.0033E+02 (4.0009E-02) =	1.0024E+02 (2.6243E-02)
f_{27}	7.8213E+02 (6.2606E+01) –	5.3680E+02 (6.5540E+01) –	1.5566E+03 (5.8870E+01) –	4.4058E+02 (5.6802E+01) –	4.5329E+02 (5.6995E+01) –	4.8675E+02 (1.2906E+02) –	3.3639E+02 (3.5793E+01)
f_{28}	1.4354E+03 (9.8737E+01) –	1.1734E+03 (4.4835E+01) –	3.8746E+02 (1.1591E+01) +	1.1289E+03 (4.1317E+01) –	1.1413E+03 (5.2616E+01) –	1.4761E+03 (1.1827E+02) –	1.1112E+03 (2.1985E+01)
f_{29}	1.4224E+03 (3.1715E+02) –	9.2726E+02 (1.3813E+02) –	2.2556E+02 (1.1299E+01) +	9.0482E+02 (1.2689E+02) –	8.9071E+02 (5.5375E+01) –	6.8024E+02 (1.2202E+02) +	8.0519E+02 (3.7243E+01)
f_{30}	1.1613E+04 (1.8321E+03) –	9.0213E+03 (5.3860E+02) –	1.0618E+03 (2.0931E+02) +	9.7197E+03 (8.1792E+02) –	9.3841E+03 (7.8714E+02) –	9.8488E+03 (5.5581E+02) –	8.7739E+03 (4.1797E+02)
w/t/l	0/4/26	2/8/20	3/6/21	0/13/17	2/14/14	3/8/19	

Table D. 3: A comparison of d DSF-EA and other algorithms on CEC'14

benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.

	SaDE	JADE	SHADE	L-SHADE	SFS	SFS-DV _{DE-GW}	d DSF-EA
f_1	9.3147E+05 (3.1419E+05) —	1.5275E+04 (1.3422E+04) —	1.8614E+04 (1.3638E+04) —	1.2000E+03 (1.5154E+03) —	1.0380E+06 (3.4650E+05) —	2.7585E+00 (5.9779E+00) +	4.0970E+02 (5.1974E+02)
f_2	3.7564E+03 (3.8115E+03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	7.9599E+03 (7.7780E+03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_3	3.4631E+03 (2.2424E+03) —	4.5134E+03 (2.3441E+03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	1.6841E+00 (2.4293E+00) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_4	8.2104E+01 (4.4460E+01) —	1.9346E+01 (3.9290E+01) +	9.7224E+00 (2.9435E+01) +	5.8866E+01 (4.5608E+01) —	9.6504E+01 (4.8574E+01) —	1.5442E+01 (3.9417E+01) +	4.6038E+01 (4.7699E+01)
f_5	2.0733E+01 (4.1549E-02) =	2.0356E+01 (3.6845E-02) =	2.0140E+01 (1.9410E-02) =	2.0249E+01 (4.5920E-02) =	2.0429E+01 (1.7556E-01) =	2.0138E+01 (5.3443E-02) =	2.0258E+01 (2.8409E-02)
f_6	1.8148E+01 (3.2634E+00) —	1.6589E+01 (6.6285E+00) —	5.1707E+00 (2.4284E+00) —	2.6408E-01 (5.2278E-01) =	2.4186E+01 (3.8939E+00) —	1.9080E-01 (4.4651E-01) +	4.2789E-01 (6.2441E-01)
f_7	1.3887E-02 (1.5352E-02) —	2.0769E-03 (4.9084E-03) —	3.9093E-03 (7.4077E-03) —	0.0000E+00 (0.0000E+00) =	1.0624E-03 (3.5592E-03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	1.0925E+00 (1.0945E+00) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	2.5817E-09 (7.4842E-09) —	6.1843E+01 (3.0497E+01) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_9	9.1673E+01 (1.4268E+01) —	5.1867E+01 (8.6673E+00) —	3.4182E+01 (6.1592E+00) —	1.1636E+01 (2.1338E+00) =	2.0178E+02 (4.0628E+01) —	3.1522E+01 (4.4965E+00) —	1.1611E+01 (2.0545E+00)
f_{10}	1.2284E+00 (1.0759E+00) —	1.2247E-02 (1.3099E-02) +	1.0287E-02 (1.1368E-02) +	1.2188E-01 (4.1286E-02) —	1.6675E+02 (1.5591E+02) —	2.2883E-03 (4.8075E-03) +	6.7828E-02 (2.6934E-02)
f_{11}	6.8299E+03 (1.5764E+03) —	3.8695E+03 (2.7785E+02) —	3.5552E+03 (3.2722E+02) —	3.2219E+03 (3.2983E+02) +	4.7934E+03 (6.9382E+02) —	3.4391E+03 (3.0948E+02) —	3.3280E+03 (3.4911E+02)
f_{12}	1.0986E+00 (1.0537E-01) —	2.5535E-01 (3.1321E-02) =	1.6366E-01 (2.1190E-02) =	2.1891E-01 (2.8176E-02) =	1.4197E-01 (8.4787E-02) =	1.6510E-01 (2.0162E-02) =	2.2020E-01 (2.6991E-02)
f_{13}	4.3681E-01 (5.4721E-02) —	3.3223E-01 (4.9590E-02) —	3.1034E-01 (4.5483E-02) —	1.6043E-01 (1.8317E-02) =	5.2075E-01 (8.3969E-02) —	2.4304E-01 (3.0417E-02) =	1.6252E-01 (2.2208E-02)
f_{14}	3.1170E-01 (3.2642E-02) =	3.1348E-01 (9.1988E-02) =	2.9453E-01 (5.7219E-02) =	3.0808E-01 (2.4695E-02) =	3.4798E-01 (1.3415E-01) =	3.2397E-01 (8.3224E-02) =	3.0843E-01 (2.8188E-02)
f_{15}	1.7092E+01 (5.8042E+00) —	7.0826E+00 (9.4236E-01) —	5.7376E+00 (6.1481E-01) —	5.2056E+00 (5.0768E-01) =	1.3024E+01 (3.2486E+00) —	6.8975E+00 (1.3995E+00) —	5.1771E+00 (4.4868E-01)
f_{16}	2.0222E+01 (2.7357E-01) —	1.7720E+01 (4.3341E-01) —	1.7444E+01 (4.2728E-01) —	1.7014E+01 (4.8120E-01) =	1.8089E+01 (7.0462E-01) —	1.7425E+01 (4.7906E-01) =	1.6997E+01 (4.1221E-01)
f_{17}	5.9118E+04 (3.9220E+04) —	2.3000E+03 (5.9502E+02) —	2.2347E+03 (7.8109E+02) —	1.4543E+03 (5.1303E+02) —	1.1219E+04 (1.0968E+04) —	1.3574E+03 (3.9909E+02) +	1.4317E+03 (3.6679E+02)
f_{18}	7.5884E+02 (5.1263E+02) —	1.8203E+02 (4.7150E+01) —	1.5375E+02 (3.6617E+01) —	1.0213E+02 (1.3841E+01) —	8.6313E+02 (1.2040E+03) —	9.9619E+01 (1.5245E+01) =	1.0014E+02 (1.3119E+01)
f_{19}	1.6741E+01 (8.7552E+00) —	1.3421E+01 (5.2199E+00) —	9.3810E+00 (3.2466E+00) —	8.2960E+00 (1.8136E+00) =	1.6699E+01 (1.3771E+01) —	1.1124E+01 (3.0602E+00) —	8.3073E+00 (1.9160E+00)
f_{20}	9.0611E+02 (7.3143E+02) —	7.9529E+03 (6.9640E+03) —	1.9816E+02 (5.7311E+01) —	1.3914E+01 (4.5644E+00) +	1.0889E+02 (3.5584E+01) —	4.7599E+01 (8.9308E+00) —	1.4126E+01 (4.3694E+00)
f_{21}	8.5745E+04	2.4322E+04	1.2297E+03	5.2593E+02	2.3415E+03	4.6959E+02	5.3548E+02

	(4.5410E+04) —	(1.6455E+05) —	(4.0604E+02) —	(1.4909E+02) +	(2.2556E+03) —	(1.4816E+02) —	(1.3736E+02)
f_{22}	5.1001E+02 (1.7759E+02) —	5.1982E+02 (1.3024E+02) —	3.6769E+02 (1.5946E+02) —	1.1716E+02 (7.5047E+01) +	6.6906E+02 (2.1832E+02) —	4.0148E+02 (1.2036E+02) —	1.2436E+02 (7.2157E+01)
f_{23}	3.4400E+02 (1.1482E-13) —	3.4400E+02 (3.7646E-13) —	3.4400E+02 (1.1482E-13) —	3.4400E+02 (4.4592E-13) —	2.0000E+02 (0.0000E+00) =	3.4400E+02 (4.5927E-13) —	2.0000E+02 (0.0000E+00)
f_{24}	2.7596E+02 (3.0905E+00) —	2.7482E+02 (1.9487E+00) —	2.7478E+02 (1.7334E+00) —	2.7522E+02 (6.6170E-01) —	2.0000E+02 (0.0000E+00) =	2.7509E+02 (8.5349E-01) —	2.0000E+02 (0.0000E+00)
f_{25}	2.1665E+02 (9.7720E+00) —	2.1699E+02 (6.7636E+00) —	2.1217E+02 (6.9165E+00) —	2.0715E+02 (3.6488E-01) —	2.0000E+02 (0.0000E+00) =	2.0546E+02 (3.2251E-01) —	2.0000E+02 (0.0000E+00)
f_{26}	1.9030E+02 (2.9941E+01) —	1.0237E+02 (1.3950E+01) —	1.0431E+02 (1.9529E+01) —	1.0212E+02 (1.3980E+01) —	1.9414E+02 (2.3667E+01) —	1.0024E+02 (2.6243E-02) —	1.0016E+02 (1.7444E-02)
f_{27}	7.8213E+02 (6.2606E+01) —	4.4058E+02 (5.6802E+01) —	4.5329E+02 (5.6995E+01) —	3.3486E+02 (3.0280E+01) —	2.0000E+02 (0.0000E+00) =	3.3639E+02 (3.5793E+01) —	2.0000E+02 (0.0000E+00)
f_{28}	1.4354E+03 (9.8737E+01) —	1.1289E+03 (4.1317E+01) —	1.1413E+03 (5.2616E+01) —	1.1124E+03 (2.9101E+01) —	2.0000E+02 (0.0000E+00) =	1.1112E+03 (2.1985E+01) —	2.0000E+02 (0.0000E+00)
f_{29}	1.4224E+03 (3.1715E+02) —	9.0482E+02 (1.2689E+02) —	8.9071E+02 (5.5375E+01) —	8.0091E+02 (2.4010E+01) —	5.0385E+02 (5.5204E+02) —	8.0519E+02 (3.7243E+01) —	3.0823E+02 (3.5311E+02)
f_{30}	1.1613E+04 (1.8321E+03) —	9.7197E+03 (8.1792E+02) —	9.3841E+03 (7.8714E+02) —	8.7599E+03 (4.1306E+02) —	8.7778E+03 (4.0000E+03) —	8.7739E+03 (4.1797E+02) —	8.7135E+03 (3.8407E+02)
w/t/l	0/2/28	2/5/23	2/6/22	4/12/14	0/8/22	6/10/14	

Table D. 4: A comparison of d DSF-EA and other algorithms on CEC'14

benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 100D averaged over 51 runs. Best entries are marked in bold.

	SaDE	JADE	SHADE	L-SHADE	SFS	SFS-DP _{DE-GW}	d DSF-EA
f_1	4.2708E+06 (8.9182E+05) —	1.1644E+05 (6.1455E+04) +	1.4280E+05 (1.0527E+05) +	1.7188E+05 (5.6543E+04) —	3.4063E+06 (8.1004E+05) —	2.0640E+05 (7.1495E+04) —	1.5481E+05 (4.4904E+04)
f_2	1.3273E+04 (9.9812E+03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	2.1043E+04 (1.5808E+04) —	2.4267E-07 (1.9604E-07) —	0.0000E+00 (0.0000E+00)
f_3	6.3462E+03 (3.7037E+03) —	5.9129E+03 (3.7758E+03) —	5.6315E-02 (2.1099E-01) —	0.0000E+00 (0.0000E+00) =	1.2151E+03 (9.4582E+02) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_4	2.1284E+02 (6.0547E+01) —	7.7261E+01 (5.5558E+01) +	9.2526E+01 (4.6796E+01) +	1.7035E+02 (3.0634E+01) —	2.3361E+02 (4.9480E+01) —	1.6701E+02 (2.7842E+01) —	1.6063E+02 (2.6181E+01)
f_5	2.1013E+01 (3.1972E-02) =	2.0482E+01 (7.4319E-02) =	2.0200E+01 (1.4669E-02) =	2.0554E+01 (3.1110E-02) =	2.0155E+01 (2.0534E-01) =	2.1285E+01 (6.5948E-02) =	2.0564E+01 (3.6034E-02)
f_6	7.5561E+01 (5.8575E+00) —	4.4605E+01 (1.4840E+01) —	3.3509E+01 (5.2490E+00) —	8.6915E+00 (2.3336E+00) +	8.2195E+01 (8.9640E+00) —	8.7679E+00 (2.2635E+00) =	9.6903E+00 (2.7112E+00)
f_7	1.1012E-02 (3.3793E-02) —	2.0760E-03 (6.0154E-03) —	1.9787E-03 (5.3391E-03) —	0.0000E+00 (0.0000E+00) +	2.0243E-03 (6.7272E-03) —	1.4502E-04 (1.0357E-03) =	1.4502E-04 (1.0357E-03)
f_8	2.0016E+01 (6.4531E+00) —	0.0000E+00 (0.0000E+00) +	0.0000E+00 (0.0000E+00) +	1.1000E-02 (7.4351E-03) —	3.8161E+02 (8.0468E+01) —	3.5982E+02 (1.1279E+01) —	4.1692E-03 (3.6511E-03)
f_9	2.9217E+02 (3.5870E+01) —	1.4425E+02 (2.0417E+01) —	1.1400E+02 (1.5505E+01) —	3.4079E+01 (5.0045E+00) +	6.4295E+02 (8.1853E+01) —	4.6490E+02 (1.7200E+01) —	3.8309E+01 (6.2988E+00)
f_{10}	3.5785E+01 (5.2364E+01) —	1.5553E-02 (9.2955E-03) +	1.0287E-02 (7.5803E-03) +	2.6429E+01 (5.7878E+00) =	1.2109E+03 (4.8627E+02) —	1.5363E+04 (5.1222E+02) —	2.5243E+01 (6.1027E+00)
f_{11}	1.2648E+04 (2.6268E+03) —	1.0730E+04 (5.4689E+02) —	9.8049E+03 (4.9663E+02) +	1.1015E+04 (5.6250E+02) —	1.2617E+04 (1.3411E+03) —	2.8312E+04 (1.3855E+03) —	1.0963E+04 (5.7809E+02)
f_{12}	1.6530E+00 (1.2847E-01) —	3.5887E-01 (1.2847E-01) =	2.2939E-01 (2.4409E-02) +	4.3723E-01 (4.6523E-02) =	1.9954E-01 (6.6936E-02) +	3.5231E+00 (6.5615E-01) —	4.5112E-01 (4.6843E-02)
f_{13}	4.9867E-01 (4.2591E-02) —	4.0910E-01 (4.0001E-02) —	3.9960E-01 (4.2796E-02) =	2.4083E-01 (2.0956E-02) =	(6.0664E-01 7.1825E-02) —	3.3446E-01 (2.9402E-02) =	2.3687E-01 (2.0001E-02)
f_{14}	1.6850E-01 (1.3967E-02) +	2.9028E+02 (2.8002E+02) —	1.2012E-01 (9.7789E-03) +	1.1781E-01 (7.3457E-03) +	3.2705E-01 (6.5182E-02) =	3.7999E-01 (1.5224E-02) =	3.3490E-01 (1.5611E-02)
f_{15}	5.8790E+01 (1.1862E+01) —	2.9730E+01 (3.9865E+00) —	2.2126E+01 (2.8739E+00) —	1.6182E+01 (1.1988E+00) =	4.7043E+01 (1.0807E+01) —	5.3599E+01 (2.1045E+00) —	1.6573E+01 (1.2374E+00)
f_{16}	4.3865E+01 (3.3444E-01) —	4.0091E+01 (5.6076E-01) —	3.9591E+01 (4.9269E-01) =	3.9146E+01 (4.7973E-01) =	4.1295E+01 (9.6696E-01) —	4.6289E+01 (2.8062E-01) —	3.9258E+01 (5.2096E-01)
f_{17}	4.2373E+05 (1.6403E+05) —	1.2039E+04 (4.8820E+03) —	1.2354E+04 (4.7972E+03) —	4.4345E+03 (7.1340E+02) =	3.9591E+05 (1.6848E+05) —	4.4248E+03 (8.0247E+02) +	4.4332E+03 (6.7563E+02)
f_{18}	7.1544E+02 (4.3296E+02) —	1.0748E+03 (9.5074E+02) —	5.0701E+02 (3.8870E+02) —	2.2364E+02 (1.6791E+01) =	2.9481E+03 (2.6760E+03) —	2.1993E+02 (1.6372E+01) =	2.2252E+02 (1.8799E+01)
f_{19}	7.0697E+01 (2.8473E+01) +	9.9450E+01 (8.3109E+00) —	9.7832E+01 (1.5491E+01) =	9.5740E+01 (2.2879E+00) =	7.4273E+01 (3.2266E+01) +	9.6397E+01 (2.6114E+00) =	9.6531E+01 (2.4790E+00)
f_{20}	6.5476E+03 (3.1507E+03) —	9.5316E+03 (1.5166E+04) —	4.9152E+02 (1.3438E+02) —	1.4992E+02 (5.2469E+01) —	7.6721E+02 (3.2262E+02) —	1.0789E+02 (2.8829E+01) +	1.3970E+02 (5.0419E+01)
f_{21}	2.9005E+05	3.6989E+03	3.6938E+03	2.3163E+03	1.5657E+05	2.0653E+03	2.2415E+03

	(1.1541E+05) —	(1.1894E+03) —	(1.6153E+03) —	(5.3084E+02) —	(7.3558E+04) —	(5.4820E+02) +	(5.2845E+02)
f_{22}	1.5644E+03 (3.6481E+02) —	1.5963E+03 (2.3319E+02) —	1.4005E+03 (2.1322E+02) —	1.1061E+03 (1.8788E+02) —	2.0722E+03 (4.5647E+02) —	2.7399E+03 (1.5752E+02) —	1.0542E+03 (1.9937E+02)
f_{23}	3.4823E+02 (1.5592E-07) —	3.4823E+02 (9.0668E-13) —	3.4823E+02 (4.5610E-13) —	3.4823E+02 (3.6628E-13) —	2.0000E+02 (0.0000E+00) =	3.4823E+02 (1.3924E-13) —	2.0000E+02 (0.0000E+00)
f_{24}	3.9636E+02 (8.1820E+00) —	3.9879E+02 (5.2302E+00) —	3.9436E+02 (4.4474E+00) —	3.9440E+02 (2.8738E+00) —	2.0000E+02 (0.0000E+00) =	3.9458E+02 (2.5541E+00) —	2.0000E+02 (0.0000E+00)
f_{25}	2.4720E+02 (1.5625E+01) —	2.7524E+02 (6.8524E+00) —	2.5836E+02 (5.8654E+00) —	2.0000E+02 (4.0892E-13) =	2.0000E+02 (0.0000E+00) =	2.0000E+02 (1.9493E-13) =	2.0000E+02 (0.0000E+00)
f_{26}	2.0016E+02 (2.3015E-02) —	2.0009E+02 (7.4145E-03) —	2.0008E+02 (1.2627E-02) —	2.0000E+02 (6.2672E-13) =	2.0000E+02 (0.0000E+00) =	2.0000E+02 (5.5088E-05) =	2.0000E+02 (0.0000E+00)
f_{27}	1.8135E+03 (1.4439E+02) —	1.0447E+03 (1.1466E+02) —	9.5033E+02 (8.8368E+01) —	3.7711E+02 (3.2793E+01) —	2.0000E+02 (0.0000E+00) =	3.9358E+02 (3.6022E+01) —	2.0000E+02 (0.0000E+00)
f_{28}	3.1541E+03 (3.4967E+02) —	2.3307E+03 (2.7172E+02) —	2.3170E+03 (2.5071E+02) —	2.3086E+03 (4.6454E+01) —	2.0000E+02 (0.0000E+00) =	2.2477E+03 (6.8746E+01) —	2.0000E+02 (0.0000E+00)
f_{29}	2.2867E+03 (3.2957E+02) —	1.3375E+03 (1.9888E+02) —	1.3451E+03 (1.4872E+02) —	7.9838E+02 (7.5828E+01) —	2.4228E+02 (3.0195E+02) —	7.5933E+02 (4.9124E+01) —	2.0000E+02 (0.0000E+00)
f_{30}	1.1546E+04 (3.2443E+03) —	8.7953E+03 (1.4283E+03) —	8.4953E+03 (9.2680E+02) —	8.3501E+03 (9.6337E+02) —	1.1499E+04 (9.3061E+03) —	8.3891E+03 (8.0731E+02) —	8.2952E+03 (7.7419E+02)
w/t/l	2/1/27	4/3/23	7/5/18	4/13/13	2/8/20	3/10/17	

Table D. 5: A comparison of LSHADE-EpSin and other algorithms on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.

	CoDE	dynNP-jDE	SaDE	JADE	SHADE	L-SHADE	LSHADE-EpSin
f_1	2.3214E+05 (1.0747E+05) —	3.2264E+05 (1.1734E+05) —	9.3147E+05 (3.1419E+05) —	1.5275E+04 (1.3422E+04) —	1.8614E+04 (1.3638E+04) —	5.4839E+02 (9.3359E+02) —	1.6673E-05 (8.4186E-05)
f_2	6.5010E+01 (1.9284E+02) —	3.3251E-07 (1.4702E-06) —	3.7564E+03 (3.8115E+03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_3	2.5387E+01 (4.1188E+01) —	7.0324E-06 (4.7809E-05) —	3.4631E+03 (2.2424E+03) —	4.5134E+03 (2.3441E+03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_4	2.6067E+01 (3.3789E+01) +	8.1654E+01 (2.1947E+01) —	8.2104E+01 (4.4460E+01) —	1.9346E+01 (3.9290E+01) +	9.7224E+00 (2.9435E+01) +	4.3999E+01 (4.7625E+01) —	5.5906E+01 (4.7552E+01)
f_5	2.0032E+01 (6.8626E-02) =	2.0384E+01 (2.9280E-02) =	2.0733E+01 (4.1549E-02) =	2.0356E+01 (3.6845E-02) =	2.0140E+01 (1.9410E-02) =	2.0262E+01 (3.0283E-02) =	2.0260E+01 (3.0625E-02)
f_6	8.4983E+00 (3.3570E+00) —	7.3352E+00 (5.6880E+00) —	1.8148E+01 (3.2634E+00) —	1.6589E+01 (6.6285E+00) —	5.1707E+00 (2.4284E+00) —	2.8208E-01 (5.2398E-01) —	1.7911E-04 (1.6705E-04)
f_7	1.5467E-03 (3.1935E-03) —	0.0000E+00 (0.0000E+00) =	1.3887E-02 (1.5352E-02) —	2.0769E-03 (4.9084E-03) —	3.9093E-03 (7.4077E-03) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	4.8773E-01 (8.2941E-01) —	0.0000E+00 (0.0000E+00) =	1.0925E+00 (1.0945E+00) —	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_9	8.2582E+01 (1.7230E+01) —	7.4286E+01 (9.3923E+00) —	9.1673E+01 (1.4268E+01) —	5.1867E+01 (8.6673E+00) —	3.4182E+01 (6.1592E+00) =	1.1732E+01 (2.5019E+00) +	3.0212E+01 (5.3749E+00)
f_{10}	5.1856E+00 (3.0648E+00) —	6.8582E-03 (9.7767E-03) +	1.2284E+00 (1.0759E+00) —	1.2247E-02 (1.3099E-02) +	1.0287E-02 (1.1368E-02) +	5.2028E-02 (2.4097E-02) =	4.9861E-02 (2.7260E-02)
f_{11}	4.3391E+03 (9.4713E+02) —	4.3297E+03 (3.8575E+02) —	6.8299E+03 (1.5764E+03) —	3.8695E+03 (2.7785E+02) —	3.5552E+03 (3.2722E+02) —	3.3013E+03 (3.3270E+02) —	3.0260E+03 (3.1360E+02)
f_{12}	8.7329E-02 (4.5691E-02) =	3.6401E-01 (4.8652E-02) =	1.0986E+00 (1.0537E-01) —	2.5535E-01 (3.1321E-02) =	1.6366E-01 (2.1190E-02) =	2.1362E-01 (2.5551E-02) =	2.1034E-01 (3.2005E-02)
f_{13}	3.3636E-01 (4.9736E-02) =	3.4874E-01 (4.1091E-02) —	4.3681E-01 (5.4721E-02) —	3.3223E-01 (4.9590E-02) —	3.1034E-01 (4.5483E-02) —	1.6635E-01 (1.7505E-02) +	2.0613E-01 (2.7022E-02)
f_{14}	2.7652E-01 (2.9839E-02) =	3.0009E-01 (2.6630E-02) —	3.1170E-01 (3.2642E-02) —	3.1348E-01 (9.1988E-02) —	2.9453E-01 (5.7219E-02) —	2.6990E-01 (1.8936E-02) —	1.9094E-01 (2.2794E-02)
f_{15}	7.4043E+00 (1.3955E+00) —	1.0077E+01 (1.1616E+00) —	1.7092E+01 (5.8042E+00) —	7.0826E+00 (9.4236E-01) —	5.7376E+00 (6.1481E-01) =	5.1011E+00 (4.1540E-01) =	5.5699E+00 (4.9977E-01)
f_{16}	1.8290E+01 (8.9113E-01) —	1.7551E+01 (4.6595E-01) =	2.0222E+01 (2.7357E-01) —	1.7720E+01 (4.3341E-01) —	1.7444E+01 (4.2728E-01) =	1.6816E+01 (4.6135E-01) =	1.6600E+01 (5.2168E-01)
f_{17}	1.5714E+04 (1.3291E+04) —	1.2583E+04 (8.3862E+03) —	5.9118E+04 (3.9220E+04) —	2.3000E+03 (5.9502E+02) —	2.2347E+03 (7.8109E+02) —	1.4640E+03 (3.6469E+02) —	3.3412E+02 (1.5841E+02)
f_{18}	3.2916E+02 (3.4839E+02) —	2.5984E+02 (2.4692E+02) —	7.5884E+02 (5.1263E+02) —	1.8203E+02 (4.7150E+01) —	1.5375E+02 (3.6617E+01) —	1.0085E+02 (1.7047E+01) —	1.9277E+01 (5.9325E+00)
f_{19}	6.1691E+00 (9.8380E-01) +	1.0832E+01 (8.4404E-01) =	1.6741E+01 (8.7552E+00) —	1.3421E+01 (5.2199E+00) —	9.3810E+00 (3.2466E+00) =	8.4924E+00 (2.0870E+00) =	9.6846E+00 (1.1238E+00)
f_{20}	2.1861E+02 (2.3515E+02) —	3.9163E+01 (1.3337E+01) —	9.0611E+02 (7.3143E+02) —	7.9529E+03 (6.9640E+03) —	1.9816E+02 (5.7311E+01) —	5.1411E+01 (2.0828E+01) —	6.0661E+00 (2.0153E+00)
f_{21}	6.4912E+03 —	2.7281E+03 —	8.5745E+04 —	2.4322E+04 —	1.2297E+03 —	6.9834E+02 —	3.1607E+02

	(5.5372E+03) —	(2.5200E+03) —	(4.5410E+04) —	(1.6455E+05) —	(4.0604E+02) —	(2.3873E+02) —	(9.9913E+01)
f_{22}	6.0223E+02 (2.2061E+02) —	4.2383E+02 (1.2826E+02) —	5.1001E+02 (1.7759E+02) —	5.1982E+02 (1.3024E+02) —	3.6769E+02 (1.5946E+02) —	1.0146E+02 (5.6017E+01) =	1.0181E+02 (5.9160E+01)
f_{23}	3.4400E+02 (1.1482E-13) —	3.4400E+02 (1.1482E-13) —	3.4400E+02 (1.1482E-13) —	3.4400E+02 (3.7646E-13) —	3.4400E+02 (1.1482E-13) —	3.4400E+02 (3.5662E-13) —	2.0000E+02 (0.0000E+00)
f_{24}	2.7139E+02 (2.9312E+00) —	2.6546E+02 (1.6705E+00) —	2.7596E+02 (3.0905E+00) —	2.7482E+02 (1.9487E+00) —	2.7478E+02 (1.7334E+00) —	2.8547E+02 (1.9879E+00) —	2.0000E+02 (8.1552E-09)
f_{25}	2.0792E+02 (4.7505E+00) —	2.0691E+02 (1.3449E+00) —	2.1665E+02 (9.7720E+00) —	2.1699E+02 (6.7636E+00) —	2.1217E+02 (6.9165E+00) —	2.0530E+02 (3.3379E-01) —	2.0000E+02 (0.0000E+00)
f_{26}	1.0618E+02 (2.3701E+01) —	1.0035E+02 (4.1976E-02) =	1.9030E+02 (2.9941E+01) —	1.0237E+02 (1.3950E+01) =	1.0431E+02 (1.9529E+01) —	1.0017E+02 (1.9297E-02) =	1.0020E+02 (3.4909E-02)
f_{27}	5.3680E+02 (6.5540E+01) —	3.8452E+02 (4.8217E+01) —	7.8213E+02 (6.2606E+01) —	4.4058E+02 (5.6802E+01) —	4.5329E+02 (5.6995E+01) —	3.0884E+02 (1.3199E+01) —	2.0196E+02 (1.4003E+01)
f_{28}	1.1734E+03 (4.4835E+01) —	1.1052E+03 (3.8147E+01) —	1.4354E+03 (9.8737E+01) —	1.1289E+03 (4.1317E+01) —	1.1413E+03 (5.2616E+01) —	9.7358E+02 (8.3564E+01) —	2.0000E+02 (0.0000E+00)
f_{29}	9.2726E+02 (1.3813E+02) —	9.9798E+02 (1.4277E+02) —	1.4224E+03 (3.1715E+02) —	9.0482E+02 (1.2689E+02) —	8.9071E+02 (5.5375E+01) —	7.4146E+02 (4.4718E+01) —	2.0000E+02 (0.0000E+00)
f_{30}	9.0213E+03 (5.3860E+02) —	8.4860E+03 (4.2315E+02) —	1.1613E+04 (1.8321E+03) —	9.7197E+03 (8.1792E+02) —	9.3841E+03 (7.8714E+02) —	1.1829E+04 (4.1923E+02) —	2.0000E+02 (0.0000E+00)
w/t/l	2/5/23	1/8/21	0/2/28	2/6/22	2/10/18	2/12/16	

Table D. 6: A comparison of EsDER-NR and other algorithms (first group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.

	CoDE	SHADE	L-SHADE	SinDE	LSHADE-EpSin	EsDER-NR
f_1	2.3214E+05 (1.0747E+05) -	1.8614E+04 (1.3638E+04) -	5.4839E+02 (9.3359E+02) -	1.0638E+07 (3.5254E+06) -	1.6673E-05 (8.4186E-05) -	5.5169E-07 (1.1974E-06)
f_2	6.5010E+01 (1.9284E+02) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	1.7998E+04 (1.8164E+04) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_3	2.5387E+01 (4.1188E+01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	1.4255E+03 (8.1830E+02) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_4	2.6067E+01 (3.3789E+01) +	9.7224E+00 (2.9435E+01) +	4.3999E+01 (4.7625E+01) =	1.6183E+02 (2.7235E+01) -	5.5906E+01 (4.7552E+01) -	4.7197E+01 (4.8486E+01)
f_5	2.0032E+01 (6.8626E-02) =	2.0140E+01 (1.9410E-02) =	2.0262E+01 (3.0283E-02) =	2.0922E+01 (6.0076E-02) =	2.0260E+01 (3.0625E-02) =	2.0245E+01 (6.0822E-02)
f_6	8.4983E+00 (3.3570E+00) -	5.1707E+00 (2.4284E+00) -	2.8208E-01 (5.2398E-01) -	1.9051E+00 (2.0964E+00) -	1.7911E-04 (1.6705E-04) +	1.2891E-02 (7.2727E-02)
f_7	1.5467E-03 (3.1935E-03) -	3.9093E-03 (7.4077E-03) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	4.8773E-01 (8.2941E-01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	4.9544E+01 (1.1072E+01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_9	8.2582E+01 (1.7230E+01) -	3.4182E+01 (6.1592E+00) -	1.1732E+01 (2.5019E+00) +	1.3634E+02 (2.3056E+01) -	3.0212E+01 (5.3749E+00) -	1.5918E+01 (3.1017E+00)
f_{10}	5.1856E+00 (3.0648E+00) -	1.0287E-02 (1.1368E-02) =	5.2028E-02 (2.4097E-02) =	4.8283E+02 (2.8244E+02) -	4.9861E-02 (2.7260E-02) =	4.9728E-02 (2.2875E-02)
f_{11}	4.3391E+03 (9.4713E+02) -	3.5552E+03 (3.2722E+02) -	3.3013E+03 (3.3270E+02) -	1.1910E+04 (1.1035E+03) -	3.0260E+03 (3.1360E+02) +	3.1201E+03 (2.8752E+02)
f_{12}	8.7329E-02 (4.5691E-02) +	1.6366E-01 (2.1190E-02) =	2.1362E-01 (2.5551E-02) =	7.8418E-01 (2.3215E-01) =	2.1034E-01 (3.2005E-02) =	2.0835E-01 (2.5431E-02)
f_{13}	3.3636E-01 (4.9736E-02) -	3.1034E-01 (4.5483E-02) -	1.6635E-01 (1.7505E-02) =	3.8002E-01 (5.2303E-02) =	2.0613E-01 (2.7022E-02) -	1.8766E-01 (2.0005E-02)
f_{14}	2.7652E-01 (2.9839E-02) =	2.9453E-01 (5.7219E-02) =	2.6990E-01 (1.8936E-02) -	2.1064E-01 (1.9458E-02) =	1.9094E-01 (2.2794E-02) =	1.8489E-01 (2.4428E-02)
f_{15}	7.4043E+00 (1.3955E+00) -	5.7376E+00 (6.1481E-01) =	5.1011E+00 (4.1540E-01) =	1.4363E+01 (2.5084E+00) -	5.5699E+00 (4.9977E-01) =	5.1590E+00 (4.7135E-01)
f_{16}	1.8290E+01 (8.9113E-01) -	1.7444E+01 (4.2728E-01) =	1.6816E+01 (4.6135E-01) =	4.0866E+01 (9.4003E-01) -	1.6600E+01 (5.2168E-01) =	1.6517E+01 (4.3527E-01)
f_{17}	1.5714E+04 (1.3291E+04) -	2.2347E+03 (7.8109E+02) -	1.4640E+03 (3.6469E+02) -	1.6875E+06 (5.5573E+05) -	3.3412E+02 (1.5841E+02) +	3.5176E+02 (1.9257E+02)
f_{18}	3.2916E+02 (3.4839E+02) -	1.5375E+02 (3.6617E+01) -	1.0085E+02 (1.7047E+01) -	8.3587E+02 (1.0310E+03) -	1.9277E+01 (5.9325E+00) -	1.3497E+01 (4.0281E+00)
f_{19}	6.1691E+00 (9.8380E-01) +	9.3810E+00 (3.2466E+00) -	8.4924E+00 (2.0870E+00) =	8.8250E+01 (1.1676E+00) -	9.6846E+00 (1.1238E+00) =	9.9508E+00 (8.4356E-01)
f_{20}	2.1861E+02 (2.3515E+02) -	1.9816E+02 (5.7311E+01) -	5.1411E+01 (2.0828E+01) =	1.9583E+03 (7.6989E+02) -	6.0661E+00 (2.0153E+00) =	5.1352E+00 (1.8296E+00)
f_{21}	6.4912E+03	1.2297E+03	6.9834E+02	1.1396E+06	3.1607E+02	3.0519E+02

	(5.5372E+03) –	(4.0604E+02) –	(2.3873E+02) –	(4.9249E+05) –	(9.9913E+01) –	(1.1219E+02)
f_{22}	6.0223E+02 (2.2061E+02) –	3.6769E+02 (1.5946E+02) –	1.0146E+02 (5.6017E+01) –	1.3503E+03 (3.7408E+02) –	1.0181E+02 (5.9160E+01) –	9.3068E+01 (5.7660E+01)
f_{23}	3.4400E+02 (1.1482E-13) –	3.4400E+02 (1.1482E-13) –	3.4400E+02 (3.5662E-13) –	3.4823E+02 (2.9112E-10) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{24}	2.7139E+02 (2.9312E+00) –	2.7478E+02 (1.7334E+00) –	2.8547E+02 (1.9879E+00) –	3.6863E+02 (3.2592E+00) –	2.0000E+02 (8.1552E-09) =	2.0000E+02 (1.6689E-11)
f_{25}	2.0792E+02 (4.7505E+00) –	2.1217E+02 (6.9165E+00) –	2.0530E+02 (3.3379E-01) –	2.4100E+02 (6.2990E+00) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{26}	1.0618E+02 (2.3701E+01) –	1.0431E+02 (1.9529E+01) –	1.0017E+02 (1.9297E-02) –	2.0075E+02 (2.1389E-01) –	1.0020E+02 (3.4909E-02) =	1.0018E+02 (1.8526E-02)
f_{27}	5.3680E+02 (6.5540E+01) –	4.5329E+02 (5.6995E+01) –	3.0884E+02 (1.3199E+01) –	3.2499E+02 (2.2350E+01) –	2.0196E+02 (1.4003E+01) =	2.0000E+02 (0.0000E+00)
f_{28}	1.1734E+03 (4.4835E+01) –	1.1413E+03 (5.2616E+01) –	9.7358E+02 (8.3564E+01) –	1.9989E+03 (1.5394E+02) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{29}	9.2726E+02 (1.3813E+02) –	8.9071E+02 (5.5375E+01) –	7.4146E+02 (4.4718E+01) –	2.2705E+03 (3.1553E+02) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{30}	9.0213E+03 (5.3860E+02) –	9.3841E+03 (7.8714E+02) –	1.1829E+04 (4.1923E+02) –	8.3783E+03 (6.2704E+02) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
w/t/l	3/2/25	1/9/20	1/13/16	0/5/25	3/20/7	

Table D. 7: A comparison of EsDER-NR and other algorithms (first group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 100D averaged over 51 runs. Best entries are marked in bold.

	CoDE	SHADE	L-SHADE	SinDE	LSHADE-EpSin	EsDER-NR
f_1	7.8408E+05 (2.3046E+05) -	1.4280E+05 (1.0527E+05) -	1.4499E+05 (4.2326E+04) -	1.0638E+07 (3.5254E+06) -	1.6397E+04 (1.1918E+04) -	4.0626E+03 (3.1112E+03)
f_2	1.2940E+04 (1.1735E+04) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	1.7998E+04 (1.8164E+04) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_3	4.8685E+02 (5.4667E+02) -	5.6315E-02 (2.1099E-01) -	0.0000E+00 (0.0000E+00) =	1.4255E+03 (8.1830E+02) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_4	1.5142E+02 (3.3456E+01) +	9.2526E+01 (4.6796E+01) +	1.5404E+02 (2.2147E+01) +	1.6183E+02 (2.7235E+01) +	1.7927E+02 (3.2404E+01) -	1.8068E+02 (3.0186E+01)
f_5	2.0028E+01 (5.4426E-02) =	2.0200E+01 (1.4669E-02) =	2.0551E+01 (3.3438E-02) =	2.0922E+01 (6.0076E-02) =	2.0547E+01 (3.9744E-02) -	2.0549E+01 (3.6060E-02)
f_6	4.4880E+01 (6.5725E+00) -	3.3509E+01 (5.2490E+00) -	8.9198E+00 (2.5608E+00) -	1.9051E+00 (2.0964E+00) -	4.9551E-01 (5.2589E-01) -	4.0122E-01 (5.0025E-01)
f_7	9.6577E-04 (3.5870E-03) -	1.9787E-03 (5.3391E-03) -	1.4502E-04 (1.0357E-03) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	1.6212E+01 (4.3565E+00) -	0.0000E+00 (0.0000E+00) +	1.3343E-03 (8.4393E-04) =	4.9544E+01 (1.1072E+01) -	2.8971E-03 (2.2582E-03) -	1.7018E-03 (1.2279E-03)
f_9	2.1084E+02 (3.3882E+01) -	1.1400E+02 (1.5505E+01) -	3.7543E+01 (4.7490E+00) -	1.3634E+02 (2.3056E+01) -	5.4639E+01 (1.3789E+01) -	3.6440E+01 (6.8210E+00)
f_{10}	8.4659E+01 (8.9816E+01) -	1.0287E-02 (7.5803E-03) +	1.7574E+01 (3.3089E+00) =	4.8283E+02 (2.8244E+02) -	1.8410E+01 (3.5915E+00) =	1.7024E+01 (3.6126E+00)
f_{11}	1.2002E+04 (1.3890E+03) -	9.8049E+03 (4.9663E+02) +	1.0579E+04 (6.6870E+02) -	1.1910E+04 (1.1035E+03) -	1.0404E+04 (5.3786E+02) -	1.0120E+04 (5.0600E+02)
f_{12}	2.8319E-01 (1.8429E-01) +	2.2939E-01 (2.4409E-02) =	4.2671E-01 (4.5726E-02) =	7.8418E-01 (2.3215E-01) =	4.2820E-01 (4.1276E-02) =	4.1063E-01 (3.3775E-02)
f_{13}	4.2927E-01 (5.7280E-02) -	3.9960E-01 (4.2796E-02) =	2.3876E-01 (1.6654E-02) =	3.8002E-01 (5.2303E-02) =	3.1769E-01 (2.5863E-02) =	2.9823E-01 (2.2943E-02)
f_{14}	1.2626E-01 (1.4810E-02) =	1.2012E-01 (9.7789E-03) =	2.2457E-01 (1.0652E-02) =	2.1064E-01 (1.9458E-02) =	1.7652E-01 (1.0707E-02) =	1.7324E-01 (1.1696E-02)
f_{15}	2.3732E+01 (4.5910E+00) -	2.2126E+01 (2.8739E+00) -	1.5824E+01 (1.3160E+00) =	1.4363E+01 (2.5084E+00) =	1.6789E+01 (8.9663E-01) =	1.5815E+01 (9.0554E-01)
f_{16}	4.2189E+01 (1.1668E+00) -	3.9591E+01 (4.9269E-01) =	3.9224E+01 (5.6571E-01) =	4.0866E+01 (9.4003E-01) -	3.8495E+01 (6.1145E-01) =	3.8261E+01 (5.2120E-01)
f_{17}	1.2409E+05 (4.0440E+04) -	1.2354E+04 (4.7972E+03) -	4.2632E+03 (8.1065E+02) -	1.6875E+06 (5.5573E+05) -	2.0153E+03 (4.9847E+02) -	1.8856E+03 (4.7923E+02)
f_{18}	9.9298E+02 (1.0156E+03) -	5.0701E+02 (3.8870E+02) -	2.2374E+02 (1.5202E+01) -	8.3587E+02 (1.0310E+03) -	9.5386E+01 (2.1281E+01) -	7.1812E+01 (1.5439E+01)
f_{19}	9.2009E+01 (1.7679E+01) =	9.7832E+01 (1.5491E+01) -	9.6138E+01 (2.6581E+00) -	8.8250E+01 (1.1676E+00) =	8.9191E+01 (1.1610E+00) =	8.8911E+01 (1.3122E+00)
f_{20}	1.9187E+03 (9.4802E+02) -	4.9152E+02 (1.3438E+02) -	1.5389E+02 (5.5748E+01) -	1.9583E+03 (7.6989E+02) -	2.1154E+01 (4.3101E+00) -	1.5933E+01 (3.0711E+00)
f_{21}	5.8445E+04 -	3.6938E+03 -	2.3112E+03 -	1.1396E+06 -	6.1724E+02 -	5.9413E+02

	(2.5505E+04) –	(1.6153E+03) –	(5.1028E+02) –	(4.9249E+05) –	(2.4514E+02) –	(2.4411E+02)
f_{22}	1.8394E+03 (3.9059E+02) –	1.4005E+03 (2.1322E+02) –	1.0482E+03 (1.7667E+02) –	1.3503E+03 (3.7408E+02) –	1.0729E+03 (2.0730E+02) –	9.7189E+02 (1.8459E+02)
f_{23}	3.4823E+02 (3.3301E-13) –	3.4823E+02 (4.5610E-13) –	3.4823E+02 (1.4648E-13) –	3.4823E+02 (2.9112E-10) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{24}	3.8531E+02 (4.7593E+00) –	3.9436E+02 (4.4474E+00) –	3.9417E+02 (2.2453E+00) –	3.6863E+02 (3.2592E+00) –	2.0000E+02 (1.1744E-09) =	2.0000E+02 (1.9551E-09)
f_{25}	2.5231E+02 (1.8111E+01) –	2.5836E+02 (5.8654E+00) –	2.0000E+02 (2.3779E-13) =	2.4100E+02 (6.2990E+00) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{26}	2.0011E+02 (1.8248E-02) =	2.0008E+02 (1.2627E-02) =	2.0000E+02 (3.7333E-13) =	2.0075E+02 (2.1389E-01) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{27}	1.1731E+03 (1.3449E+02) –	9.5033E+02 (8.8368E+01) –	3.8770E+02 (3.6763E+01) –	3.2499E+02 (2.2350E+01) –	2.0000E+02 (1.9773E-04) =	2.0000E+02 (2.0000E+02)
f_{28}	2.3304E+03 (2.4505E+02) –	2.3170E+03 (2.5071E+02) –	2.2498E+03 (5.0263E+01) –	1.9989E+03 (1.5394E+02) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{29}	1.5277E+03 (1.7129E+02) –	1.3451E+03 (1.4872E+02) –	7.5818E+02 (5.4654E+01) –	2.2705E+03 (3.1553E+02) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
f_{30}	8.6182E+03 (9.8207E+02) –	8.4953E+03 (9.2680E+02) –	7.9530E+03 (9.7862E+02) –	8.3783E+03 (6.2704E+02) –	2.0000E+02 (0.0000E+00) =	2.0000E+02 (0.0000E+00)
w/t/l	2/4/23	4/7/19	1/13/16	1/7/22	0/18/12	

Table D. 8: A comparison of EsDER-NR and other algorithms (second group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 50D averaged over 51 runs. Best entries are marked in bold.

	GaAPADE	MVMO	UMOEAsII	iCMAES-ILS	NBIPOP-aCMA-ES	EsDER-NR
f_1	1.1728E+03 (3.7791E+03) -	7.5393E-03 (1.6192E-03) -	1.8340E-03 (1.1487E-03) -	0.0000E+00 (0.0000E+00) +	0.0000E+00 (0.0000E+00) +	5.5169E-07 (1.1974E-06)
f_2	0.0000E+00 (0.0000E+00) =	1.5210E-05 (5.9390E-06) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_3	0.0000E+00 (0.0000E+00) =	4.5420E-03 (6.4863E-04) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_4	4.9494E+01 (4.9078E+01) -	5.7708E+00 (2.3313E+01) -	3.8489E+01 (4.8395E+01) +	1.0321E+01 (2.4404E+01) +	0.0000E+00 (0.0000E+00) +	4.7197E+01 (4.8486E+01)
f_5	2.0006E+01 (5.9577E-03) =	2.0000E+01 (5.2143E-05) =	2.0000E+01 (1.9808E-04) =	2.0000E+01 (6.4343E-05) =	2.0775E+01 (5.1913E-01) =	2.0245E+01 (6.0822E-02)
f_6	1.0381E+01 (3.5001E+00) -	1.0514E+01 (3.8081E+00) -	2.0301E-01 (4.5025E-01) -	2.2410E-04 (9.5551E-04) +	1.3083E+00 (2.2075E+00) -	1.2891E-02 (7.2727E-02)
f_7	1.9328E-04 (1.3803E-03) -	6.6684E-03 (5.4482E-03) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	0.0000E+00 (0.0000E+00) =	2.2826E+00 (1.5831E+00) -	0.0000E+00 (0.0000E+00) =	5.7596E+00 (2.0063E+00) -	2.5230E+01 (8.8365E+00) -	0.0000E+00 (0.0000E+00)
f_9	3.3362E+01 (8.6993E+00) -	7.6931E+01 (1.8508E+01) -	4.1004E+00 (1.9463E+00) +	6.3691E+00 (1.8704E+00) +	5.8184E+00 (2.9572E+00) +	1.5918E+01 (3.1017E+00)
f_{10}	3.4478E-01 (3.6958E-01) -	8.0772E+01 (8.9427E+01) -	5.4314E-01 (6.3474E-01) -	1.6002E+02 (2.5156E+02) -	2.0431E+03 (7.7407E+02) -	4.9728E-02 (2.2875E-02)
f_{11}	4.2372E+03 (5.4006E+02) -	3.5713E+03 (5.5154E+02) -	3.4703E+03 (4.8248E+02) -	1.1285E+02 (1.2668E+02) +	1.6667E+03 (8.7696E+02) +	3.1201E+03 (2.8752E+02)
f_{12}	1.4684E-01 (8.6748E-02) =	4.9289E-02 (4.1327E-02) +	1.3033E-01 (7.1472E-02) =	1.0269E-02 (3.0335E-02) +	1.3074E-02 (6.8484E-03) -	2.0835E-01 (2.5431E-02)
f_{13}	2.0989E-01 (4.5348E-02) =	2.8443E-01 (3.8757E-02) -	1.4660E-01 (3.8445E-02) =	6.5999E-02 (1.8572E-02) +	7.9652E-02 (2.9484E-02) +	1.8766E-01 (2.0005E-02)
f_{14}	2.5953E-01 (3.1832E-02) =	2.3750E-01 (2.8826E-02) -	3.0411E-01 (2.2844E-02) -	2.4268E-01 (3.2539E-02) -	3.1757E-01 (5.5584E-02) =	1.8489E-01 (2.4428E-02)
f_{15}	6.7704E+00 (7.4234E-01) =	4.9989E+00 (1.0527E+00) -	5.2876E+00 (8.1693E-01) =	4.5639E+00 (4.0711E-01) -	4.4424E+00 (5.5478E-01) =	5.1590E+00 (4.7135E-01)
f_{16}	1.8676E+01 (6.1719E-01) -	1.8860E+01 (4.8007E-01) -	1.8264E+01 (7.3921E-01) -	1.9934E+01 (7.2026E-01) -	1.8940E+01 (1.7918E+00) -	1.6517E+01 (4.3527E-01)
f_{17}	1.0322E+03 (4.0737E+02) -	6.6477E+02 (3.2215E+02) -	1.0892E+03 (3.4469E+02) -	2.1217E+03 (4.0225E+02) -	1.8479E+03 (2.8355E+02) -	3.5176E+02 (1.9257E+02)
f_{18}	4.6955E+01 (1.9380E+01) -	3.4011E+01 (2.9658E+01) -	7.2763E+01 (1.7400E+01) -	1.7181E+02 (8.2964E+01) -	2.4651E+02 (4.8358E+01) -	1.3497E+01 (4.0281E+00)
f_{19}	9.0591E+00 (2.3999E+00) =	6.7581E+00 (1.1646E+00) +	8.7867E+00 (1.9786E+00) =	1.4430E+01 (2.2866E+00) -	1.2386E+01 (2.1240E+00) -	8.7508E+00 (8.4356E-01)
f_{20}	3.1197E+01 (1.7026E+01) -	2.2867E+01 (7.1048E+00) -	1.3626E+01 (3.9552E+00) -	2.9297E+02 (8.5036E+01) -	2.4263E+02 (6.7944E+01) -	5.1352E+00 (1.8296E+00)

f_{21}	5.4510E+02 (2.0042E+02) –	5.3883E+02 (2.0070E+02) –	4.5210E+02 (1.3276E+02) –	1.5441E+03 (3.3065E+02) –	1.4456E+03 (3.0574E+02) –	3.0519E+02 (1.1219E+02)
f_{22}	3.7624E+02 (1.1410E+02) –	4.5841E+02 (1.4391E+02) –	1.9825E+02 (9.6317E+01) –	2.0142E+02 (1.6253E+02) –	2.7080E+02 (1.3871E+02) –	9.3068E+01 (5.7660E+01)
f_{23}	3.4400E+02 (0.0000E+00) –	3.4400E+02 (4.5927E-13) –	2.0000E+02 (0.0000E+00) =	3.4400E+02 (5.5818E-11) –	3.4400E+02 (1.1482E-13) –	2.0000E+02 (0.0000E+00)
f_{24}	2.7373E+02 (2.1945E+00) –	2.5788E+02 (3.6554E+00) –	2.0000E+02 (0.0000E+00) =	2.6243E+02 (2.8619E+00) –	2.7123E+02 (2.6252E+00) –	2.0000E+02 (1.6689E-11)
f_{25}	2.0530E+02 (4.0734E-01) =	2.0617E+02 (1.6048E+00) –	2.0000E+02 (0.0000E+00) =	2.0533E+02 (3.0463E-01) =	2.0503E+02 (2.7028E-01) =	2.0000E+02 (0.0000E+00)
f_{26}	1.0017E+02 (3.2929E-02) =	1.0415E+02 (1.9587E+01) =	1.0014E+02 (3.5609E-02) =	1.0018E+02 (1.3186E-01) =	1.0020E+02 (1.0146E-01) =	1.0018E+02 (1.8526E-02)
f_{27}	5.6604E+02 (5.7387E+01) –	5.1076E+02 (7.7736E+01) –	2.0000E+02 (0.0000E+00) =	3.1017E+02 (1.7298E+01) –	3.1745E+02 (2.2268E+01) –	2.0743E+02 (3.0371E+01)
f_{28}	1.1615E+03 (3.6653E+01) –	1.3267E+03 (6.2978E+01) –	2.0000E+02 (0.0000E+00) =	1.2465E+03 (7.9789E+01) –	1.1495E+03 (7.5408E+01) –	2.0000E+02 (0.0000E+00)
f_{29}	3.2603E+07 (9.6122E+06) –	1.1056E+03 (2.3285E+02) –	8.0845E+02 (4.6427E+01) –	8.0538E+02 (4.7359E+01) –	7.9378E+02 (2.9560E+01) –	2.0000E+02 (0.0000E+00)
f_{30}	8.9317E+03 (5.4645E+02) –	9.4213E+03 (3.5442E+02) –	8.6188E+03 (4.6476E+02) –	9.4846E+03 (6.0962E+02) –	1.0412E+04 (6.3779E+02) –	2.0000E+02 (0.0000E+00)
$w/t/l$	0/11/19	2/2/26	2/14/14	5/6/17	6/7/17	

Table D. 9: A comparison of EsDER-NR and other algorithms (second group) on CEC'14 benchmark problems. Mean and standard deviation for functions (f_1 - f_{30}) on 100D averaged over 51 runs. Best entries are marked in bold.

	GaAPADE	MVMO	UMOEAsII	iCMAES-ILS	NBIPOP-aCMA-ES	EsDER-NR
f_1	1.5337E+05 (6.9922E+04) -	4.8207E-03 (4.3227E-04) +	3.9337E-03 (6.2611E-04) +	0.0000E+00 (0.0000E+00) +	0.0000E+00 (0.0000E+00) +	4.0626E+03 (3.1112E+03)
f_2	3.3542E-08 (4.2502E-08) =	1.0646E-04 (1.4310E-04) -	3.4889E-09 (2.0974E-08) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_3	0.0000E+00 (0.0000E+00) =	2.0197E-02 (3.6331E-03) -	1.2672E-09 (4.4281E-09) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_4	1.5948E+02 (3.0777E+01) +	3.4930E+01 (5.7275E+01) +	1.6244E+02 (4.4306E+01) +	1.1399E+02 (4.7370E+01) +	3.7364E+01 (4.8442E+01) +	1.8068E+02 (3.0186E+01)
f_5	2.0011E+01 (1.5886E-02) =	2.0000E+01 (1.6600E-05) =	2.0000E+01 (1.2317E-04) =	2.0000E+01 (1.6745E-05) =	2.1234E+01 (3.1224E-01) =	2.0549E+01 (3.6060E-02)
f_6	4.4779E+01 (1.0268E+01) -	4.5703E+01 (1.0688E+01) -	9.7992E+00 (3.2518E+00) +	7.1187E-02 (2.2388E-01) +	5.3821E+00 (5.2727E+00) -	4.0122E-01 (5.0025E-01)
f_7	6.2805E-04 (2.5530E-03) -	5.2606E-03 (8.3365E-03) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	5.0333E+00 (2.9345E+01) -	1.5205E+01 (4.2538E+00) -	0.0000E+00 (0.0000E+00) =	1.9372E+01 (3.0528E+00) -	4.5584E+01 (1.3904E+01) -	1.7018E-03 (1.2279E-03)
f_9	1.7248E+02 (7.5987E+01) -	1.9165E+02 (3.8831E+01) -	2.5206E+01 (4.1772E+00) +	2.0076E+01 (3.3036E+00) +	6.9306E+00 (3.1730E+00) +	3.8440E+01 (6.8210E+00)
f_{10}	7.1026E+01 (2.7072E+02) -	1.2970E+03 (5.5624E+02) -	7.9435E+01 (9.7727E+01) -	9.3122E+02 (6.5079E+02) -	5.1602E+03 (2.5302E+03) -	1.7024E+01 (3.6126E+00)
f_{11}	1.1201E+04 (8.4866E+02) -	1.0037E+04 (5.6696E+02) +	1.1254E+04 (9.2042E+02) -	1.5072E+03 (7.9557E+02) +	3.7184E+03 (1.7075E+03) +	1.0120E+04 (5.0600E+02)
f_{12}	2.4759E-01 (6.5390E-02) -	5.7555E-02 (5.3995E-02) +	2.6369E-01 (1.1881E-01) =	6.6211E-04 (5.9429E-04) +	1.1177E-02 (5.1605E-03) +	4.1063E-01 (3.3775E-02)
f_{13}	4.1652E-01 (8.5310E-02) =	3.2764E-01 (3.0694E-02) =	1.9318E-01 (5.7856E-02) +	2.3937E-01 (3.1799E-02) =	2.7032E-01 (6.8737E-02) =	2.9823E-01 (2.2943E-02)
f_{14}	2.0684E-01 (1.7839E-02) =	1.7880E-01 (1.2803E-02) =	2.1913E-01 (1.3148E-02) =	1.3519E-01 (1.6261E-02) =	1.1829E-01 (1.3506E-02) =	1.7324E-01 (1.1696E-02)
f_{15}	1.6311E+01 (2.8812E+00) =	1.6026E+01 (2.7258E+00) -	1.2567E+01 (1.8034E+00) =	9.7916E+00 (8.3677E-01) +	9.6826E+00 (1.0018E+00) +	1.5815E+01 (9.0554E-01)
f_{16}	4.1523E+01 (8.2907E-01) =	4.1664E+01 (5.4032E-01) =	4.1727E+01 (1.4552E+00) =	4.2481E+01 (7.6764E-01) =	4.2894E+01 (2.0001E+00) =	3.8261E+01 (5.2120E-01)
f_{17}	3.8822E+03 (5.8167E+02) -	3.0945E+03 (4.2474E+02) -	4.2924E+03 (.3538E+02) -	5.0697E+03 (7.4660E+02) -	4.8426E+03 (6.2393E+02) -	1.8856E+03 (4.7923E+02)
f_{18}	3.7024E+02 (6.8761E+02) -	8.0273E+01 (1.6196E+01) -	2.1755E+02 (1.4132E+01) -	5.4302E+02 (1.6865E+02) -	6.5953E+02 (9.8763E+01) -	7.1812E+01 (1.5439E+01)
f_{19}	9.4355E+01 (1.9937E+00) -	9.5306E+01 (2.2421E+01) -	9.5545E+01 (2.3057E+00) -	8.2081E+01 (3.3548E+01) =	1.0186E+02 (1.6098E+01) -	8.8911E+01 (1.3122E+00)
f_{20}	1.7836E+02 (4.4764E+01) -	7.3234E+01 (1.7522E+01) -	1.3503E+02 (3.4112E+01) -	6.5283E+02 (1.2415E+02) -	6.1615E+02 (1.2175E+02) -	1.5933E+01 (3.0711E+00)

f_{21}	1.7311E+03 (5.0012E+02) –	1.3958E+03 (3.8070E+02) –	1.9720E+03 (4.4631E+02) –	3.3163E+03 (7.3945E+02) –	3.3947E+03 (6.6601E+02) –	5.9413E+02 (2.4411E+02)
f_{22}	1.2705E+03 (2.2906E+02) –	1.3305E+03 (2.5549E+02) –	1.1129E+03 (3.2001E+02) –	5.3750E+02 (1.7846E+02) +	7.6199E+02 (2.4961E+02) +	9.7189E+02 (1.8459E+02)
f_{23}	3.4823E+02 (5.4853E-13) –	3.4823E+02 (5.7409E-14) –	2.0000E+02 (0.0000E+00) =	3.4823E+02 (9.2710E-09) –	3.4823E+02 (5.2837E-13) –	2.0000E+02 (0.0000E+00)
f_{24}	3.8755E+02 (3.0567E+00) –	3.6069E+02 (2.6651E+00) –	2.0000E+02 (0.0000E+00) =	3.5287E+02 (9.4871E+00) –	3.8826E+02 (2.6101E+00) –	2.0000E+02 (1.9551E-09)
f_{25}	2.0055E+02 (3.9079E+00) –	2.5201E+02 (1.0940E+01) –	2.0000E+02 (0.0000E+00) =	2.1786E+02 (1.8559E+00) –	2.1757E+02 (2.1945E+00) –	2.0000E+02 (0.0000E+00)
f_{26}	1.8829E+02 (3.2400E+01) –	1.9631E+02 (1.9604E+01) –	2.0000E+02 (0.0000E+00) =	1.9118E+02 (2.7156E+01) –	1.4930E+02 (5.0279E+01) –	2.0000E+02 (0.0000E+00)
f_{27}	1.2183E+03 (3.0437E+02) –	9.5830E+02 (1.2613E+02) –	2.0000E+02 (0.0000E+00) =	3.0004E+02 (1.9298E-01) –	3.0041E+02 (2.7678E+00) –	2.0000E+02 (2.0000E+02)
f_{28}	2.3507E+03 (1.8030E+02) –	2.6866E+03 (3.4739E+02) –	2.0000E+02 (0.0000E+00) =	2.4871E+03 (8.6743E+01) –	2.4552E+03 (3.0507E+02) –	2.0000E+02 (0.0000E+00)
f_{29}	8.2536E+07 (1.1891E+07) –	2.1724E+03 (3.2615E+02) –	8.8793E+02 (1.6958E+02) –	1.5076E+03 (7.1816E+02) –	7.5387E+02 (3.0314E+01) –	2.0000E+02 (0.0000E+00)
f_{30}	7.6258E+03 (1.4421E+03) –	8.5221E+03 (1.4256E+03) –	8.0936E+03 (8.5819E+02) –	8.7123E+03 (1.3161E+03) –	9.0424E+03 (1.7524E+03) –	2.0000E+02 (0.0000E+00)
w/t/l	1/7/22	4/4/28	4/15/11	8/8/14	7/7/16	

Table D. 10: A comparison of **L-covnSHADE** and other algorithms on CEC'11

real-world problems. Mean and standard deviation for each problem. Best entries are marked in bold.

	GA-MPC	SFS	EA-DE-MA	SAMODE	LSHADE	L-covnSHADE
T01	9.9922E-01 (3.5228E+00) —	1.2823E+00 (3.5918E+0) —	2.0949E+00 (4.3064E+0) —	1.2120E+00 (3.3762E+0) —	3.2449E-08 (9.4125E-08) =	1.0603E-06 (3.0248E-06) =
T02	-2.7142E+01 (8.4287E-01) =	-2.6921E+01 (8.3124E-01) =	-2.5899E+01 (2.2425E+0) =	-2.7070E+01 (6.6248E-01) =	-2.6180E+01 (3.5847E-01) =	-2.6103E+01 (4.0487E-01) =
T03	1.1515E-05 (5.6761E-19) =	1.1515E-05 (5.4276E-19) =	1.1515E-05 (9.9711E-19) =	1.1515E-05 (0.0000E+0) =	1.1515E-05 (2.4203E-19) =	1.1515E-05 (2.2096E-19) =
T04	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+0) =	1.4369E+01 (9.9597E-01) —	1.3941E+01 (2.5022E-01) —	0.0000E+00 (0.0000E+0) =	0.0000E+00 (0.0000E+00) =
T05	-3.3949E+01 (1.4708E+00) —	-3.4603E+01 (1.7754E+0) —	-3.6309E+01 (7.9161E-01) =	-3.3595E+01 (1.5751E+0) —	-3.6340E+01 (6.1395E-01) =	-3.6165E+01 (7.0262E-01) =
T06	-2.7138E+01 (2.4443E+00) —	-2.7208E+01 (2.0222E+0) —	-2.7336E+01 (2.6153E+0) —	-2.7635E+01 (1.9235E+0) —	-2.9153E+01 (2.2185E-02) =	-2.9151E+01 (1.0448E-02) =
T07	2.0590E+00 (4.3477E-02) —	9.9690E-01 (2.4282E-01) +	5.2784E-01 (3.5525E-02) +	8.1662E-01 (1.1937E-01) +	1.2166E+00 (7.5432E-02) =	1.1223E+00 (7.4856E-02) =
T08	2.2000E+02 (0.0000E+00) =	2.2000E+02 (0.0000E+0) =	2.2000E+02 (0.0000E+0) =	2.2000E+02 (0.0000E+0) =	2.2000E+02 (0.0000E+0) =	2.2000E+02 (0.0000E+00) =
T09	2.2664E+03 (9.8854E+02) +	2.6308E+03 (6.7090E+0) +	5.2214E+02 (9.1986E+0) +	2.2644E+03 (4.7704E+0) +	8.1688E+03 (4.8923E+03) +	2.7681E+04 (1.9528E+04) =
T10	-2.1657E+01 (1.2697E-01) =	-2.1702E+01 (1.4573E-01) =	-2.1255E+01 (9.0483E-01) =	-2.1659E+01 (1.1296E-01) =	-2.1548E+01 (1.0392E-01) =	-2.1587E+01 (1.0338E-01) =
T10.1	5.3814E+04 (8.2216E+03) —	5.3285E+04 (7.0721E+0) —	1.5157E+06 (7.1759E+0) —	5.2348E+04 (2.0317E+0) —	5.2040E+04 (4.4232E+02) —	5.1867E+04 (5.1534E+02) =
T10.2	1.0776E+06 (1.1922E+04) —	1.0835E+06 (1.2830E+0) —	2.0598E+07 (6.0596E+0) —	1.0731E+06 (1.2727E+0) —	1.0705E+06 (1.4597E+03) +	1.0711E+06 (1.5777E+03) =
T11.1	1.5444E+04 (8.7481E-06) =	1.5444E+04 (2.1080E-03) =	1.5451E+04 (4.2415E+0) =	1.5444E+04 (3.7333E-02) =	1.5444E+04 (4.5835E-06) =	1.5444E+04 (2.6679E-06) =
T11.2	1.8332E+04 (8.4991E+01) —	1.8396E+04 (1.0044E+0) —	1.9205E+04 (1.2423E+0) —	1.8525E+04 (1.0714E+0) —	1.8097E+04 (3.1064E+01) —	1.8090E+04 (2.6695E+01) =
T11.3	3.2795E+04 (3.8600E+01) —	3.2878E+04 (5.1940E+0) —	3.2909E+04 (4.9102E+0) —	3.2846E+04 (7.7779E+0) —	3.2740E+04 (2.4250E-01) =	3.2740E+04 (9.9485E-02) =
T11.4	1.3644E+05 (2.7429E+03) —	1.2808E+05 (1.8826E+0) —	3.5305E+05 (6.0933E+0) —	1.3271E+05 (8.0726E+0) —	1.2391E+05 (5.2353E+02) —	1.2374E+05 (5.8388E+02) =
T11.5	2.1709E+06 (4.8034E+05) —	1.9193E+06 (1.1641E+0) —	2.5836E+06 (6.4696E+0) —	1.9771E+06 (1.2557E+0) —	1.8581E+06 (1.3225E+04) —	1.8457E+06 (1.2742E+04) =
T12.1	1.0278E+06 (1.6472E+05) —	9.4064E+05 (1.6930E+0) —	1.6638E+06 (3.2774E+0) —	9.4892E+05 (6.7159E+0) —	9.3323E+05 (1.4983E+03) —	9.3319E+05 (1.4198E+03) =
T12.2	1.3025E+06 (1.9915E+05) —	1.0187E+06 (5.7952E+0) —	2.3514E+06 (4.5468E+0) —	1.2059E+06 (7.2980E+0) —	9.4022E+05 (9.2320E+02) —	9.3973E+05 (1.3008E+03) =
T12.3	1.0382E+06 (4.2769E+04) —	9.4252E+05 (3.0640E+0) —	1.6638E+06 (3.2774E+0) —	9.5879E+05 (7.6155E+0) —	9.3200E+05 (1.2615E+03) —	9.3119E+05 (1.4254E+03) =
	1.2956E+01	1.2826E+01	1.3768E+01	1.1067E+01	1.5475E+01	1.5245E+01

T13	(3.4119E+00) +	(3.6367E+0) +	(2.3739E+0) +	(2.6523E+0) +	(9.4831E-01) =	(1.2198E+00)
T14	1.0515E+01 (2.1345E+00) =	1.2608E+01 (3.5148E+0) -	1.7386E+01 (3.6891E+0) -	1.0995E+01 (2.3890E+0) =	1.1264E+01 (2.5719E+00) =	1.1300E+01 (2.6588E+00)
w/t/1	2/7/13	3/6/13	3/6/13	2/6/14	2/13/7	

Table D. 11: A comparison of **iDEaSm** and other algorithms. Mean and standard deviation for functions (f_1 - f_{15}) on 10D. Best entries are marked in bold.

	JADE	CoDE	SaDE	EPSDE	ESMDE	L-SHADE	iDEaSm
f_1	1.1500E-43 (3.2600E-43) -	1.4400E-58 (4.5200E-58) -	2.4800E-101 (1.2000E-100) -	1.6800E-98 (4.5100E-98) -	2.9852E-80 (2.3327E-80) -	3.7870E-114 (2.0459E-113) -	2.9475E-170 (0.0000E+00)
f_2	1.2600E+00 (2.1400E+00) -	1.7300E-26 (4.6700E-26) -	1.33E-01 (7.2800E-15) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_3	3.5500E-15 (0.0000E+00) =	3.4300E-15 (6.4900E-16) =	1.3000E-15 (1.7400E-15) =	1.4200E-15 (1.7700E-15) =	3.5527E-15 (0.0000E+00) =	3.3159E-15 (9.0135E-16) =	3.0790E-15 (1.2283E-15)
f_4	5.5000E-12 (1.2100E-11) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	2.5400E-04 (1.3500E-03) -	0.0000E+00 (0.0000E+00) =	7.4528E-02 (1.7456E-02) -	0.0000E+00 (0.0000E+00)
f_5	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_6	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_7	3.9500E+00 (2.1600E+01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	7.9000E+00 (3.0100E+01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	2.8500E-59 (1.4800E-58) -	1.6300E-34 (3.5400E-34) -	8.7200E-77 (4.7700E-76) -	3.3700E-74 (4.8800E-74) -	6.5437E-38 (1.2576E-37) -	5.3009E-85 (1.7796E-84) -	1.3879E-118 (4.1995E-118)
f_9	1.3000E-53 (2.3000E-53) -	3.4000E-30 (1.0400E-29) -	1.1500E-64 (4.7400E-64) -	1.0500E-65 (3.0500E-65) -	1.1369E-12 (8.1414E-13) -	4.1457E-67 (2.2546E-66) -	1.6758E-85 (6.6526E-85)
f_{10}	7.0700E-39 (1.9600E-38) -	3.9400E-56 (5.2700E-56) -	5.6000E-99 (1.5000E-98) -	2.2100E-95 (5.1100E-95) -	2.3384E-76 (2.0280E-76) -	3.0307E-110 (1.5482E-109) -	8.8838E-167 (0.0000E+00)
f_{11}	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_{12}	3.6800E-22 (6.4800E-22) -	1.4700E-31 (1.1200E-31) -	4.0300E-52 (6.8400E-52) -	4.1300E-49 (8.9400E-49) -	2.1398E-40 (1.0714E-40) -	1.9355E-56 (3.1379E-56) -	1.0719E-78 (5.5385E-78)
f_{13}	2.8800E-24 (4.9000E-24) -	5.3300E-17 (6.5300E-17) -	3.87E-27 (7.5200E-27) -	7.9300E-22 (6.9000E-22) -	4.6566E-18 (2.3130E-18) -	8.2963E-41 (2.7014E-40) -	1.0887E-57 (3.2918E-57)
f_{14}	1.5700E-32 (5.5700E-48) =	1.5700E-32 (5.5700E-48) =	1.5700E-32 (5.5700E-48) =	1.5700E-32 (5.5700E-48) =	1.5705E-32 (5.5674E-48) =	1.5705E-32 (5.5674E-48) =	1.5705E-32 (5.5674E-48)
f_{15}	1.3500E-32 (5.5700E-48) =	1.3500E-32 (5.5700E-48) =	1.3500E-32 (5.5700E-48) =	1.3500E-32 (5.5700E-48) =	1.3498E-32 (5.5674E-48) =	1.3498E-32 (5.5674E-48) =	1.3498E-32 (5.5674E-48)
w/t/l	0/6/9	1/6/8	1/6/8	0/7/8	0/8/7	0/8/7	

Table D. 12: A comparison of **iDEaSm** and other algorithms for bound-constrained benchmark set. Mean and standard deviation for functions (f_1 - f_{15}) on $30D$. Best entries are marked in bold.

	JADE	CoDE	SaDE	EPSDE	ESMDE	L-SHADE	iDEaSm
f_1	1.2127E-22 (1.2367E-22) -	7.2000E-61 (1.7200E-60) -	2.8646E-87 (3.1584E-87) -	4.2600E-122 (7.6100E-122) =	1.3254E-88 (9.2662E-89) -	3.0515E-94 (6.3433E-94) -	6.7602E-121 (1.3266E-119)
f_2	5.3100E-01 (1.3800E+00) -	1.1500E-07 (3.8900E-07) -	9.3000E-01 (1.7200E+00) -	3.9900E-01 (1.2200E+00) -	1.6713E-01 (1.1763E-01) -	1.5062E-24 (1.3560E-25) -	3.5206E-26 (1.7179E-25)
f_3	3.5500E-15 (0.0000E+00) =	3.5500E-15 (0.0000E+00) =	3.5500E-15 (0.0000E+00) =	3.5500E-15 (0.0000E+00) =	4.8554E-15 (1.7413E-15) =	3.5527E-15 (0.0000E+00) =	3.5527E-15 (0.0000E+00)
f_4	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	3.5300E-03 (7.0100E-03) -	9.0400E-04 (2.7800E-03) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_5	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	1.1250E-15 (1.3587E-15) -	0.0000E+00 (0.0000E+00)
f_6	0.0000E+00 (0.0000E+00) =	3.4500E-05 (1.0600E-04) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_7	3.9500E+00 (2.1600E+01) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	1.5900E+02 (1.4100E+02) -	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00)
f_8	7.9961E-22 (1.1517E-21) -	1.8900E-16 (2.8100E-16) -	6.8500E-22 (2.9200E-21) -	2.8800E-29 (1.1500E-28) -	9.9716E-01 (2.8631E-01) -	2.0052E-43 (4.9213E-43) =	3.9952E-43 (2.9215E-42)
f_9	3.0500E-14 (1.5900E-13) -	1.0600E-03 (2.3100E-03) -	6.6800E-01 (1.2200E+00) -	1.1900E-08 (3.2900E-08) -	7.8741E+01 (2.3448E+01) -	2.7561E-26 (8.4671E-25) =	9.4197E-26 (2.6766E-25)
f_{10}	8.3900E-108 (4.6000E-107) -	3.9100E-58 (8.7300E-58) -	3.4801E-84 (3.2820E-84) -	9.4100E-118 (4.7000E-117) =	2.0878E-84 (1.4929E-84) -	3.9670E-89 (1.3037E-88) -	2.0254E-114 (3.5294E-115)
f_{11}	0.0000E+00 (0.0000E+00) =	0.0000E+00 (0.0000E+00) =	4.1300E-03 (1.4700E-02) -	0.0000E+00 (0.0000E+00) -	0.0000E+00 (0.0000E+00) -	0.0000E+00 (0.0000E+00) -	0.0000E+00 (0.0000E+00)
f_{12}	4.3200E-31 (1.5800E-30) -	4.9300E-31 (6.9000E-31) -	4.1505E-52 (2.1042E-52) =	1.2800E-62 (1.3200E-62) +	4.2896E-44 (1.7398E-44) -	1.3151E-43 (6.7087E-43) -	1.1997E-52 (1.6926E-52)
f_{13}	2.1237E-09 (1.2216E-09) -	5.4700E-11 (3.7500E-11) -	4.4200E-23 (1.5100E-22) -	5.7400E-13 (4.6200E-13) -	4.8086E-08 (1.5683E-08) -	2.0321E-23 (4.1249E-23) -	8.1536E-26 (1.7526E-25)
f_{14}	1.5700E-32 (5.5700E-48) =	1.5700E-32 (5.5700E-48) =	1.5700E-32 (5.5800E-48) =	1.5700E-32 (5.5700E-48) =	1.5705E-32 (5.5674E-48) =	1.5705E-32 (5.5674E-48) =	1.5705E-32 (5.5674E-48)
f_{15}	1.3500E-32 (5.5700E-48) =	1.3500E-32 (5.5700E-48) =	7.3330E-04 (2.7900E-03) -	1.3500E-32 (5.5700E-48) =	1.3498E-32 (5.5674E-48) =	1.3498E-32 (5.5674E-48) =	1.3498E-32 (5.5674E-48)
w/t/l	0/7/8	0/7/8	0/5/11	1/7/8	0/6/9	0/8/7	

Table D. 13: A comparison of **iDEaSm** and other algorithms on CEC'11 real-world problems. Mean and standard deviation for each problem. Best entries are marked in bold.

	EA-DE-MA	ENSMML_DE	WL_DE	DE-RHC	GA-MPC	L-SHADE	iDEaSm
T1	7.6023E+00 (6.0206E+00) -	7.0600E+00 (6.0400E+00) -	3.5400E+00 (5.9300E+00) -	1.2600E+01 (6.0900E+00) -	1.2678E+00 (4.3986E+00) -	2.8408E+00 (5.1638E+00) -	7.5494E-10 (2.0371E-09)
T2	-2.3020E+01 (3.5638E+00) -	-1.2600E+01 (1.8900E+00) -	-1.8300E+01 (3.7800E+00) -	-2.3900E+01 (1.9500E+00) -	-1.1477E+01 (3.1872E+00) -	-2.1001E+01 (1.7039E+00) -	-2.7401E+01 (6.4837E-01)
T5	-3.5204E+01 (1.5347E+00) =	-2.3800E+01 (2.0500E+00) -	-3.2000E+01 (3.2600E+00) -	-3.5200E+01 (1.1100E+00) =	-2.6377E+01 (6.3227E+00) -	-3.3703E+01 (4.0358E-01) -	-3.5547E+01 (1.0027E+00)
T6	2.4833E+01 (3.0506E+00) -	-1.2800E+01 (1.9800E+00) -	-2.6200E+01 (2.7800E+00) =	-2.8600E+01 (1.4100E+00) +	-1.7332E+01 (3.6484E+00) -	-2.6875E+01 (2.1270E+00) =	-2.2454E+01 (3.5641E+00)
T7	5.8474E-01 (7.7935E-02) =	1.6100E+00 (7.7400E-02) -	1.0000E+00 (1.7300E-01) -	1.3100E+00 (1.1700E-01) -	1.6135E+00 (3.7303E-01) -	1.2778E+00 (5.1890E-02) -	7.6208E-01 (1.2079E-01)
T8	2.2000E+02 (0.0000E+00) =	2.2000E+02 (0.0000E+00) =	2.2000E+02 (0.0000E+00) =	2.2000E+02 (0.0000E+00) =	2.2000E+02 (0.0000E+00) =	2.2000E+02 (0.0000E+00) =	2.2000E+02 (0.0000E+00)
T10	-1.8949E+01 (3.2443E+00) -	-1.5200E+01 (3.5900E+00) -	-2.1600E+01 (1.9400E-01) =	-1.5500E+01 (2.3000E+00) -	-2.1629E+01 (1.2381E-01) =	-2.1462E+01 (8.4907E-02) =	-2.1508E+01 (1.3122E-01)
T14	1.9297E+04 (1.8436E+02) -	1.8200E+04 (7.7300E+01) -	1.8400E+04 (1.0200E+02) -	1.9300E+04 (1.9100E+02) -	1.8386E+04 (1.5209E+02) -	1.8131E+04 (6.0957E+01) -	1.8118E+04 (5.0147E+01)
T15	3.3022E+04 (7.1048E+01) -	3.2800E+04 (5.2500E+01) -	3.2800E+04 (6.2300E+01) -	3.3100E+04 (7.5300E+01) -	3.2887E+04 (4.6263E+01) -	3.2765E+04 (4.0029E+01) -	3.2752E+04 (5.9879E+00)
T16	3.5790E+05 (9.6664E+03) -	1.3200E+05 (2.8300E+03) -	1.4000E+05 (3.9000E+03) -	1.3900E+05 (3.2600E+03) -	1.4129E+05 (2.4745E+03) -	1.2703E+05 (2.3027E+03) -	1.2606E+05 (5.3807E+02)
T21	1.6232E+01 (2.9456E+00) -	2.2600E+01 (1.9100E+00) -	1.4200E+01 (3.0100E+00) =	2.0200E+01 (2.3300E+00) -	1.8219E+01 (5.5009E+00) -	1.7658E+01 (2.0562E+00) -	1.5458E+01 (1.7363E+00)
T22	1.9708E+01 (4.8234E+00) -	1.9900E+01 (5.9600E+00) -	1.6600E+01 (4.4200E+00) =	2.0900E+01 (3.1600E+00) -	1.6034E+01 (2.9284E+00) =	1.8981E+01 (3.1157E+00) -	1.7737E+01 (2.4031E+00)
w/t/l	0/3/9	0/1/11	0/5/7	1/2/9	0/3/9	0/3/9	

Table D. 14: A comparison of **iDEaSm** and other algorithms for welded beam design problem

Methods	Optimal design variables				Best	Worst	Mean	Std	FEs
	x_1	x_2	x_3	x_4					
Tomassetti [139]	0.205729	3.470489	9.036624	0.205730	1.7248	N.A	N.A	N.A	10,000
Kaveh & Talatahari [140]	0.207301	3.435699	9.041934	0.205714	1.723377	1.762567	1.743454	0.007356	4,000
Deb [136]	0.248900	6.173000	8.178900	0.253300	2.433116	N.A	N.A	N.A	N.A
Aragon et al. [141]	0.244369	6.218613	8.291474	0.244369	2.38113	2.710406	2.439811	0.093146	320,000
Coello & Montes [142]	0.205986	3.471328	9.020224	0.206480	1.728226	1.792654	1.993408	0.074713	N.A
Salimi [86]	0.2057296	3.470488	9.0366239	0.205729	1.7248523	1.7248523	1.7248523	7.7087E-16	50,000
iDEaSm	0.20572963	3.4704888	9.0366238	0.20572965	1.7248523	1.7248864	1.7248564	7.6933E-06	4,425

* N.A: Not Available

Table D. 15: A comparison of **iDEaSm** and other algorithms for the pressure vessel design problem

Methods	Optimal design variables				Best	Worst	Mean	Std	FEs
	x_1	x_2	x_3	x_4					
Sandgren [137]	1.1250	0.6250	48.3807	11.7449	8048.6190	N.A	N.A	N.A	N.A
Huang et. al. [143]	0.8125	0.4375	42.0984	176.7465	6059.7340	6085.2303	6371.0455	43.0130	27,500
Krohling & Coelho [144]	0.8125	0.4375	42.0913	176.7465	6061.0777	6147.1332	6363.8041	86.4500	240,000
Coello & Montes [142]	0.8125	0.4375	42.0973	176.65405	6059.9463	6177.2533	6469.3220	130.9297	80,000
Gandomi [138]	0.8125	0.4375	42.09844	176.63659	6059.71433	6495.3470	6447.7360	502.693	N.A
Salimi [86]	0.8125	0.4375	42.098445	176.63659	6059.71433	6059.71433	6059.71433	6059.71433	50,000
iDEaSm	0.778198	0.384665	40.321054	199.98023	5988.027532	5890.400037	5887.10836	1.82062166	20,000

* N.A: Not Available