Novel Strategies for Single and Multi-Objective Imperialistic Competitive Algorithm

Zhavat Sherinov

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Eastern Mediterranean University
January 2018
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Assoc. Prof. Dr. Ali Hakan Ulusoy
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Asst. Prof. Dr. Ahmet Ünveren
Supervisor

Examining Committee

1. Prof. Dr. Tolga Çiloğlu _____

2. Prof. Dr. Kemal Leblebicioğlu _____

3. Assoc. Prof. Dr. Mehmet Bodur _____

4. Asst. Prof. Dr. Adnan Acan _____

5. Asst. Prof. Dr. Ahmet Ünveren _____

ABSTRACT

In this thesis, two different algorithms for solving global optimization problems were developed. The first is imperialistic competitive algorithm with updated assimilation (ICAMA), which is used for solving single-objective optimization problems. ICAMA is a new strategic improvement on the imperialist competitive algorithm (ICA) that is originally proposed based on inspirations from imperialistic competition. Another algorithm is a multi-objective imperialistic competitive algorithm (MOICA), which is for global multi-objective optimization problems.

ICA is based on the idea of imperialism. Two fundamental components of ICA are empires and colonies. Initially, the algorithm builds several randomly initialized empires where each empire includes one emperor and several colonies. Competitions take place between the empires and these competitions result in the development of more powerful empires and the collapse of the weaker ones. In ICAMA a new method is introduced for the movement of colonies towards their imperialist, which is called assimilation. The proposed method uses Euclidean distance along with Pearson correlation coefficient as an operator for assimilating colonies with respect to their imperialists. In order to test the effectiveness and competitiveness of ICAMA against other state of the art algorithms it was applied to three sets of benchmark problems – the set of 23 classical benchmark problems, CEC2005 and CEC2015 benchmarks.

MOICA is a modified multi-objective version of ICA. MOICA incorporates the competition between empires and their colonies for the solution of multi-objective

problems. Therefore, it employs a proposed approach of several non-dominated solution sets, whereby each set is called a local non-dominated solution set (LNDS). All imperialists in an empire are considered non-dominated solutions, whereas all colonies are considered dominated solutions. Aside from local non-dominated solution sets, there is one global non-dominated solution set (GNDS), which is created from LNDS sets of all empires. MOICA is applied to a number of benchmark problems such as the set of ZDT problems and CEC2009 multi-objective optimization benchmark problems set.

Simulations and experimental results on the benchmark problems showed that ICAMA produces competitive results for many test problems compared to other state-of-the-art algorithms used in this study. Moreover, MOICA is more efficient with comparison to many of the competitor algorithms used in this study, since it produces better results for most of the test problems.

Keywords: Multi-objective metaheuristics, imperialistic competitive algorithm, multiple non-dominated sets, global optimization.

ÖZ

Bu tezde, tümel optimizasyon problemlerini çözmek için iki farklı algoritma geliştirilmiştir. Birincisi, gerçek değerli tek amaçlı en iyileme problemlerinin çözümü için geliştirilmiş asimilasyon operatörü ile emperyalist rekabet algoritmasıdır (ICAMA). ICAMA emperyalist rekabetten gelen ilhamlara dayanarak emperyalist rekabetçi algoritmada (ICA) yeni bir stratejik gelişmedir. Diğeri ise, çok amaçlı global optimizasyon problemler için geliştirilmiş olan çok amaçlı bir emperyalist rekabet algoritmasıdır (MOICA).

ICA, emperyalizm fikrine dayanıyor. ICA'nın iki temel bileşeni imparatorluklar ve kolonilerdir. Başlangıçta, algoritma her imparatorluğun bir imparator ve birkaç koloni içerdiği birkaç rasgele başlatılmış imparatorluklar oluşturur. İmparatorluklar arasında yarışmalar yapılır ve bu yarışmalar daha güçlü imparatorlukların gelişmesine ve daha zayıf olanların çökmesine neden olur. ICAMA'da kolonilerin emperyalistlerine doğru asimilasyon hareketi için yeni bir yöntem geliştirildi. Önerilen yöntem, Kolonileri emperyalistlerine göre asimile etmek için bir operatör olarak Pearson korelasyon katsayısı ile birlikte Öklid uzaklıklarını kullanmaktadır. ICAMA'nın etkililiğini ve rekabet gücünü farklı ve yeni algoritmalara karşı test etmek için üç kriter sorunu setine uygulandı – 23 standart ölçüt problemi olan seti, CEC2005 ve CEC2015.

MOICA ICA'nın değiştirilmiş çok amaçlı versiyonudur. MOICA, çok amaçlı problemlerin çözümü için imparatorluklar ve sömürgeler arasındaki rekabeti içeriyor. Bu amaçla, hakim olan birçok çözüm setinin önerilen bir yaklaşımı uygulanmaktadır

ve her bir sete yerel hakim olan çözüm seti (LNDS) adı verilmiştir. Bir imparatorluktaki tüm emperyalistler hakim olan çözümler olarak görülürken, tüm koloniler baskın çözümler olarak kabul edilir. Yerel hakim olan çözüm setlerinin yanı sıra, tüm imparatorlukların LNDS setlerinden oluşan bir tane global hakim olan çözüm seti (GNDS) vardır. MOICA, ZDT problemleri seti ve CEC2009 çok amaçlı optimizasyon kriter problem seti gibi bir dizi kriter problemine uygulanmaktadır.

Çok amaçlı problemler üzerindeki simülasyonlar ve deney sonuçları mevcut büyük, tek ve çok amaçlı optimizasyon algoritmalarına göre ICAMA ve MOICA'nın birçok test problemi için rekabetçi sonuçlar ürettikleri ve daha verimli oldukları görülmüştür.

Kıyaslama problemlerinde simülasyonlar ve deney sonuçları, ICAMA'nın bu tezde kullanılan diğer yeni algoritmalara kıyasla birçok test problemi için rekabetçi sonuçlar verdiğini gösterdi. Dahası, MOICA, bu tezde kullanılan en yeni yarışmacıların çoğunluğuna kıyasla daha verimli, çünkü test problemlerinin çoğunda daha iyi sonuçlar üretiyor.

Anahtar Kelimeler: Çok amaçlı metaheuristik, emperyalist rekabetçi algoritma, çoklu hâkim olan setler, global optimizasyon.

To My Family

ACKNOWLEDGMENT

First of all, I would like to say Alhamdulillah for giving me the strength and determination in completing this thesis throughout many years. I Praise Allah alone for His mercy and help. I am very grateful to my parents, my mother Duriya Sherinova and my father Alipasha Sherinov, for their invaluable support, patience and believe in me and who sacrificed everything for me and my studies.

My sincere thanks go to my supervisor Asst. Prof. Dr. Ahmet Ünveren for his assistance, direction and guidance. In particular, his recommendations and suggestions have been invaluable for the thesis and our publications. I also wish to thank Asst. Prof. Dr. Adnan Acan and Assoc. Prof. Dr. Mehmet Bodur for their beneficial advices and corrections they used to make during semester progress report presentations.

I am particularly indebted to my wife Amina for her invaluable patience, support and advices. I am thankful to her for restricting herself in many things during the time I was busy with my studies. I am also very grateful to my daughter Malika and my son Mahdi, for they were reasons for me to be in mood and happy.

Lastly, but by no means least, my thanks go to my sisters Sevgi and Seylan and other family members and my friends.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ v

DEDICATION vii

ACKNOWLEDGMENT viii

LIST OF TABLES xii

LIST OF FIGURES xv

LIST OF SYMBOLS AND ABBREVIATIONS xvi

1 INTRODUCTION 1

2 SINGLE AND MULTI-OBJECTIVE OPTIMIZATION PROBLEMS 5

2.1 Single-objective Optimization Problem 5

2.1.1 Formulation of Single-objective Optimization Problem 5

2.2 Multi-objective Optimization Problem 6

2.2.1 Formulation of Multi-objective Optimization 7

2.2.2 Pareto Dominance in Multi-objective Optimization 7

3 IMPERIALISTIC COMPETITIVE ALGORITHM AND ITS APPLICATIONS . 10

3.1 Literature Review 10

3.2 Review of ICA 15

3.2.1 Assimilation 16

3.2.2 Revolution 17

3.2.3 Imperialistic Competition 18

3.3 Application of Modified ICA for the Solution of Travelling Salesman Problem (TSP) 20

3.3.1 Formulation of TSP 20

3.3.2 Modified ICA 21
3.4 Experimental Results 23
4 IMPERIALISTIC COMPETITIVE ALGORITHM WITH UPDATED ASSIMILATION FOR SOLVING SINGLE-OBJECTIVE OPTIMIZATION PROBLEMS 26
4.1 The Proposed Assimilation Strategy 26
4.2 Experimental Results 30
4.2.1 Experimental Evaluations with Classical Benchmark Problems 30
4.2.2 Experimental Evaluations with CEC2015 Benchmark Problems 36
4.2.3 Experimental Analysis on the Strategy of Parameter *v* 42
5 MULTI-OBJECTIVE IMPERIALISTIC COMPETITIVE ALGORITHM WITH MULTIPLE NON-DOMINATED SETS FOR THE SOLUTION OF GLOBAL OPTIMIZATION PROBLEMS 44
5.1 Literature Review 45
5.2 Overview of MOICA 51
5.2.1 Non-Domination Sorting 55
5.2.2 Assimilation 56
5.2.3 Revolution 58
5.2.4 Possessing an Empire 60
5.2.5 Uniting Similar Empires 60
5.2.6 Imperialistic Competition 62
5.2.7 Computational Complexity of MOICA 64
5.3 Experimental Results 64
5.3.1 Discussion of the Results 65
5.3.2 Friedman Aligned Ranks Test 79
6 CONCLUSION 80

REFERENCES..81

LIST OF TABLES

Table 3.1. Obtained best and average results of ICA for different instances..... 25

Table 4.1. Experimental results for 23 classical benchmark problems..... 32

Table 4.2. Best found results for unimodal functions..... 33

Table 4.3. Best found results for multimodal functions..... 33

Table 4.4. Best found results for multimodal functions with a few local minima..... 34

Table 4.5. Friedman aligned ranks..... 34

Table 4.6. Friedman aligned ranks statistics. 35

Table 4.7. Best, worst, mean and standard deviation scores achieved by ICAMA for the 15 CEC2015 competition benchmark problems with dimension of 30 36

Table 4.8. Best, worst, mean and standard deviation scores achieved by ICAMA for the 15 CEC2015 competition benchmark problems with dimension of 10 37

Table 4.9. Mean results for function 1 of CEC2015 competition from best to worst with dimension sizes of 10 and 30..... 37

Table 4.10. Mean results for function 2 of CEC2015 competition from best to worst with dimension sizes of 10 and 30..... 37

Table 4.11. Mean results for function 3 of CEC2015 competition from best to worst with dimension sizes of 10 and 30..... 38

Table 4.12. Mean results for function 4 of CEC2015 competition from best to worst with dimension sizes of 10 and 30..... 38

Table 4.13. Mean results for function 5 of CEC2015 competition from best to worst with dimension sizes of 10 and 30..... 38

Table 4.14. Mean results for function 6 of CEC2015 competition from best to worst with dimension sizes of 10 and 30..... 39

Table 4.15. Mean results for function 7 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 39
Table 4.16. Mean results for function 8 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 39
Table 4.17. Mean results for function 9 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 40
Table 4.18. Mean results for function 10 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 40
Table 4.19. Mean results for function 11 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 41
Table 4.20. Mean results for function 12 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 41
Table 4.21. Mean results for function 13 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 41
Table 4.22. Mean results for function 14 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 42
Table 4.23. Mean results for function 15 of CEC2015 competition from best to worst with dimension sizes of 10 and 30 42
Table 4.24. The results of ICAMA on CEC2015 problems for different values of *v* 43
Table 5.1. Unconstrained test problems used in this study 67
Table 5.2. *Hypervolume* results for unconstrained test problems with 25,000 function evaluations 68
Table 5.3. *Epsilon Indicator* results for unconstrained test problems with 25,000 function evaluations 68

Table 5.4. *IGD* results for unconstrained test problems with 25,000 function evaluations 68

Table 5.5. *IGD* results for ZDT test problems with 25,000 function evaluations 69

Table 5.6. *Hypervolume* results for CEC 2009 unconstrained test problems with 25,000 function evaluations 72

Table 5.7. *Epsilon Indicator* results for CEC 2009 unconstrained test problems with 25,000 function evaluations 73

Table 5.8. *IGD* results for CEC 2009 unconstrained test problems with 25,000 function evaluations 73

Table 5.9. *IGD* results for CEC 2009 unconstrained test problems with 25,000 function evaluations 73

Table 5.10. *Hypervolume* results for CEC 2009 unconstrained test problems with 5,000 function evaluations 75

Table 5.11. *Epsilon Indicator* results for CEC 2009 unconstrained test problems with 5,000 function evaluations 75

Table 5.12. *IGD* results for CEC 2009 unconstrained test problems with 5,000 function evaluations 76

Table 5.13. *IGD* results for CEC 2009 unconstrained test problems with 25,000 function evaluations 76

Table 5.14. Ranking of MOICA compared to algorithms in CEC 2009 77

Table 5.15. Friedman aligned ranks over CEC2009 UF problem instances 79

LIST OF FIGURES

Figure 2.1. Dominated area for Pareto dominance 8

Figure 2.2. Decision space vs. objective space 9

Figure 2.3. Pareto optimal solutions 9

Figure 3.1. Moving a colony towards its relevant imperialist in a randomly deviated direction [2] 16

Figure 3.2. Demonstration of the fragmentation method for revolution process 22

Figure 3.3. Obtained optimal result for berlin52.tsp instance with cost = 7542 24

Figure 3.4. Obtained result for eil101.tsp with cost = 650 24

Figure 4.1. ICAMA algorithm flowchart 29

Figure 5.1. GNDS and LNDS sets of three empires 53

Figure 5.2. Non-dominancy using fronts 56

Figure 5.3. Assimilation of a colony towards randomly selected imperialist from the GNDS set 57

Figure 5.4. Generational distance for uniting empires 61

Figure 5.5. MOICA flowchart 63

Figure 5.6. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on Schaffer 70

Figure 5.7. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on ZDT4 71

Figure 5.8. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on ZDT6 72

Figure 5.9. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on UF10 75

LIST OF SYMBOLS AND ABBREVIATIONS

Ω	Some universe
a_r	Assimilation Rate
α	Revolution Rate
ξ	Replacement Rate
γ	A parameter that Adjusts the Deviation from the Original Direction
Θ	A Random Variable with Uniform Distribution Between $(-\gamma, \gamma)$
β	A Fixed Algorithmic Parameter with Value of About Two
ε	A Constant Parameter
v	A Constant Parameter
p_e	Probability for *Economic Changes*
ϕ	Maximum Percentage of Imperialists in an Empire
ABC	Artificial Bee Colony algorithm
ACO	Ant Colony Optimization
AMGA	Archive-based Micro Genetic Algorithm
CCP	Control Chart Pattern
CICA	Chaotic Improved ICA
CEC	Congress on Evolutionary Computation
CE_k	Cost of the Colonies of the Empire k
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CMA-ES_QR	A variant of CMA-ES for Expensive Scenarios
DE	Differential Evolution

DECMOSA-SQP	Differential Evolution with Self-Adaptation and Local Search for Constrained Multi-Objective Optimization Algorithm
DMCMOABC	Dynamic Multi-Colony Multi-Objective Artificial Bee Colony Algorithm
DMOEA	Dynamic Multi-Objective Optimization Algorithm
DMOEA-DD	DMOEA with Domain Decomposition
EA	Evolutionary Algorithms
ED	Euclidean Distance
EI	Epsilon Indicator
EP	Evolutionary Programming
ES	Evolutionary Strategies
FACTS	Flexible Alternating Current Transmission System
FAR	Friedman Aligned Ranks
GA	Genetic Algorithms
GD	Generational Distance
GDE3	Generalized Differential Evolution 3
GNDS	Global Non-Dominated Solutions
HBMO	Honey Bee Mating Optimization
HGA	Hybrid of Genetic Algorithms
HIM	Hybrid Intelligent Method
HumanCog	Algorithm based on Human Cognitive Behavior
HV	Hypervolume
ICA	Imperialistic Competitive Algorithm
ICA-ANN	ICA and Neural Network
IC_k	Imperialist Cost of the Empire *k*

ICAMA	Imperialistic Competitive Algorithm with Modified Assimilation
IGD	Inverted Generational Distance
iSRPSO	improved Self Regulating Particle Swarm Optimization
K-MICA	K-means ICA
LNDS	Local Non-Dominated Solutions
MICA	Modified Imperialist Competitive Algorithm
MLP	Multi-layer Perceptron
MOICA	Multi-objective Imperialistic Competitive Algorithm
MOEA	Multi-Objective Evolutionary Algorithm
MOEAD	Multi-Objective Evolutionary Algorithm based on Decomposition
MOEA/D-AWA	MOEAD with Adaptive Weight Vector Adjustment
MOEADGM	MOEAD with Guided Mutation
MOEP	Multi-Objective Evolutionary Programming
MOO	Multi-Objective Optimization
MOP	Multi-objective Optimization Problem
MTS	Multiple Trajectory Search
MVMO	Mean-Variance Mapping Optimization
NP-hard	Non-deterministic Polynomial-time hard
NSGA-II	Non-dominated Sorting Genetic Algorithm II
NSGAIILS	NSGA-II with an Augmented Local Search
NTC_k	Normalized Total Cost of the Empire *k*
OMOEAII	Orthogonal Multi-Objective Evolutionary Algorithm II
OMOPSO	Optimized Multi-Objective PSO

OWMOSaDE	Objective-Wise Multi-Objective SaDE Algorithm
Pcc	Pearson Correlation Coefficient
PSO	Particle Swarm Optimization
SA	Simulated Annealing
SaDE	Self-adaptive Differential Evolution Algorithm
SPEA	Strength Pareto Evolutionary Algorithm
SPEA2	Strength Pareto Evolutionary Algorithm 2
TC_k	Total Cost of the Empire k
TS	Tabu Search
TSP	Travelling Salesman Problem
TSPLIB	Travelling Salesman Problem Library
TunedCMAES	Parameter tuned CMA-ES for Expensive Problems

Chapter 1

INTRODUCTION

Evolutionary Algorithms (EA) have been rapidly developed in the recent decades and there are many various algorithms among EAs that address real world problems. EAs are the heuristics that are inspired by the natural processes and are applied in various fields for such tasks as optimization and searching. Genetic Algorithms (GA), Evolutionary Programming (EP) and Evolutionary Strategies (ES) are the well-known types of Evolutionary Algorithms, which have a population over which certain operators are applied in order to find the best solution. However, real life problems are often concerned about more than one objective to be optimized and often with conflicting objective functions where one should be minimized, while the other one to be maximized [1].

Multi-objective optimization problems (MOP) involve finding several optimal solutions, which are called non-dominated solutions, where each solution is known as *Pareto optimum*, while in the objective space altogether they represent a *Pareto front*. The main goal of MOPs is to find Pareto front, which requires an algorithm a significant amount of time for exploration of a search space as it is usually large and evaluating objective functions. Multi-objective Evolutionary Algorithms (MOEA) became very popular in the mid of 1990s, when more researchers were attracted by them. These days there are various applications of MOEAs almost in all fields. Some

well-known evolutionary algorithms for solving MOPs are NSGA-II [51], SPEA2 [66], MOEAD [67], etc.

In this thesis, Imperialistic Competitive Algorithm (ICA) [2] was analyzed and studied in details. ICA is an algorithm inspired by imperialistic competition between empires in which socio-political characteristics and assimilations occur during evolution process. ICA was applied on Travelling Salesman Problem [63]. However, since ICA is originally designed for real valued problems, in order to apply it on TSP it had to be modified for the solution of TSP, which is an integer valued problem. Application of ICA on TSP showed that it performs well not only for real valued benchmarks, but also for such NP-hard problems as TSP. Many benchmark instances from TSPLIB were used for testing ICA, where it could reach an optimal solution for some benchmark instances.

Further studies were carried out for the improvement of ICA. By modifying assimilation operator, which is one of the two operators of ICA, a new and improved version with modified assimilation (ICAMA) was developed [95]. While assimilating colonies towards their respective imperialist, ICAMA uses either the Euclidean distance or element-wise difference like in the original ICA, and this decision is made randomly. Moreover, modified assimilation also uses a Pearson correlation coefficient, which is computed from the newly generated colony and the imperialist, as an acceptance criterion of the newly generated colony. ICAMA performed better than original ICA and other state-of-the-art algorithms for various benchmark instances as illustrated in experimental sections.

Finally, a multi-objective version of ICA (MOICA) [96] was developed, which is another and the most important work in this thesis. MOICA, unlike many other multi-objective evolutionary algorithms, has its population divided into imperialists and colonies. MOICA implements the idea of imperialism by incorporating the competition between the empires. Every empire has a set of imperialists and a set of colonies. The main novelty of this algorithm lies in using a non-dominated solution set for every empire, where each such set is called a local non-dominated solution set. Therefore, all imperialists in an empire are considered to be non-dominated solutions, whereas all colonies are considered to be dominated solutions. Moreover, aside from local non-dominated solution sets, there is one global non-dominated solution set, which is created from all local non-dominated solution sets of all empires. Two main operators of the proposed algorithm – Assimilation and Revolution use global and local non-dominated solution sets respectively during assimilation and revolution of colonies. The significance of this study is seen from the competitive results produced by the proposed algorithm. Another significant feature in the proposed algorithm is that no special parameter is used for diversity preservation, which enables algorithm to avoid extra computations in order to maintain spread of solutions. Therefore, the proposed algorithm with original operators Assimilation and Revolution produces competitive results with comparison to the state-of-the-art-algorithms used in this study.

The organization of other chapters in this thesis is as follows. Chapter 2 briefly discusses the multi-objective optimization and its problems. Chapter 3 focuses on the overview of ICA and its applications with obtained experimental results. In Chapter 4, ICA with updated assimilation for solving single-objective optimization problems along with experimental results is discussed. Chapter 5 presents a detailed

description of a novel multi-objective version of ICA and provides experimental results conducted in this study with various benchmark problems. Finally, the conclusion and discussion of possible future work is presented in Chapter 6.

Chapter 2

SINGLE AND MULTI-OBJECTIVE OPTIMIZATION PROBLEMS

2.1 Single-objective Optimization Problem

There are various types of problems in the world that need to be optimized with respect to a single as well as multiple objectives. Single-objective optimization occurs in situations when there is only one objective to be optimized, i.e. minimized or maximized, while this objective is subject to some constraints. Examples of single-objective optimization problem may include the following: minimizing the distance travelled, maximizing the profit, maximizing the customer satisfaction, maximizing load capacity of vehicles, etc.

2.1.1 Formulation of Single-objective Optimization Problem

As was mentioned above, single-objective optimization problem has one objective to be optimized. The mathematical formulation of a single-objective optimization problem can be formulated as follows:

minimizing (*or maximizing*) *f(x)*

Subject to *k* inequality constraints:

$$g_i(x) \leq 0 \qquad i = 1, 2, \ldots, k$$

And *m* equality constraints:

$$h_i(x) = 0 \qquad i = 1, 2, \ldots, m$$

where $\boldsymbol{x}$ in Ω

A solution minimizes (or maximizes) the scalar *f(x)* where x is an *n*-dimensional decision variable vector $x = (x_1, x_2, \ldots, x_n)$ from some universe Ω.

Even though some real world problems can be expressed in a form of a single objective problem very often it is hard to define all the aspects in terms of a single objective. Therefore, defining several objectives often produces a better solution to the problem.

2.2 Multi-objective Optimization Problem

Multi-objective optimization, which is also called multi-criteria or multi-attribute optimization, is applied on the problems that involve several objective functions to be optimized simultaneously, where these objective functions are subject to some constraints. Multi-objective optimization problems (MOP) exist in many fields, such as industry, mathematics and engineering. Often in these problems there are conflicting objective functions, since some objectives need to be minimized, while others to be maximized. MOP in its nature does not have a single solution unlike single-objective optimization problems, but rather it has a set of solutions, which are called Pareto optimal solutions. Some of the examples with conflicting objectives for which MOP is solved can be as follows: minimizing the distance travelled by a vehicle and maximizing the number of customers served, minimizing the number of employees and maximizing the productivity performance, minimizing vehicles waiting times and maximizing customer satisfaction grade, etc.

2.2.1 Formulation of Multi-objective Optimization

As was mentioned above, MOP has many objective functions to be optimized which are subject to some constraints. The mathematical formulation of MOP can be formulated as follows:

$$(\max or) \min[\, f_1(x),\, f_2(x),...,\, f_n(x)] \tag{2.1}$$

Subject to *k* inequality constraints:

$$g_i(x) \le 0 \qquad i = 1, 2,\ldots, k \tag{2.2}$$

And *m* equality constraints:

$$h_i(x) = 0 \qquad i = 1, 2,\ldots, m \tag{2.3}$$

Where $n \ge 2$ is the number of objective functions to be optimized and *x* is the feasible set of decision variables, which is defined as follows: $x = [x_1, x_2,..., x_n,]^T$. The feasible set is typically defined by some constraint functions. Thus, with feasible set of values $-x*_1, x*_2,..., x^*_n$, we aim to find the optimal solutions by minimizing (or maximizing) objective functions in (2.1) and at the same time satisfying constraints in (2.2) and (2.3).

2.2.2 Pareto Dominance in Multi-objective Optimization

Since the MOP has several objective functions to be optimized it is not possible to simple pick up the best solution among the available ones, since there is a set of so called best solutions – Pareto optimal solutions. Therefore, the solution for finding Pareto optimal solutions the *dominancy* is used, which divides all solutions in a population into two groups – dominated and *non-dominated* solutions. Non-dominated set of solutions is actually the Pareto optimal set. Therefore, they are

chosen as the best found solutions for the given problem. The dominancy rule is stated as follows: *A state **A** dominates a state **B**, if **A** is better than **B** in at least one objective function and not worse with respect to all other objective functions.*

Mathematically, *Pareto dominance* is described as follows: A vector $\vec{u} = (u_1,..., u_k)$ is said to dominate $\vec{v} = (v_1,..., v_k)$ (denoted by $\vec{u} \preceq \vec{v}$) if and only if *u* is partially less than *v*, such that $\forall i \in \{1,..., k\}, u_i \leq v_i \wedge \exists i \in \{1,...k\} : u_i < v_i$.

Figure 2.1 illustrates the Pareto dominance for two of possible cases – minimization and maximization of two objective functions. In case of minimization of objective functions, solutions that are on the upper right side area are the dominated solutions. On the other hand, in case of maximization of objective functions, solutions that are on the left bottom side area become the dominated solutions of a solution ***A*** as depicted in Figure 2.1.

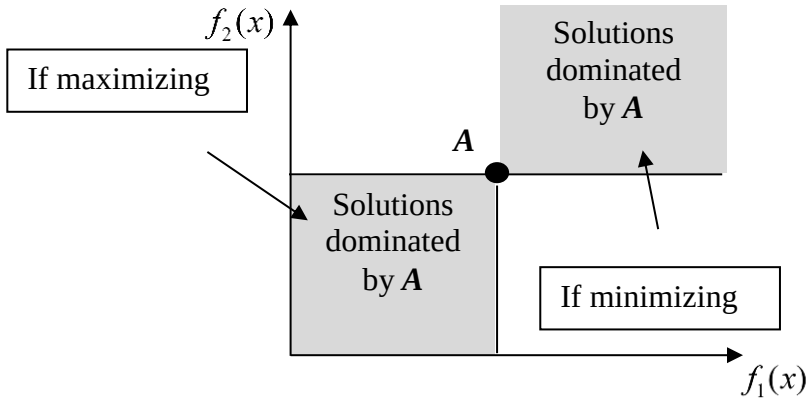


Figure 2.1. Dominated area for Pareto dominance

In multi-objective optimization the objective functions constitute a multi-dimensional space. Therefore, for each solution **x** in the decision variable space *X* there is a point in the decision space *Z* denoted by $f(x) = z = (z_1, z_2, \dots, z_m,)^T$.

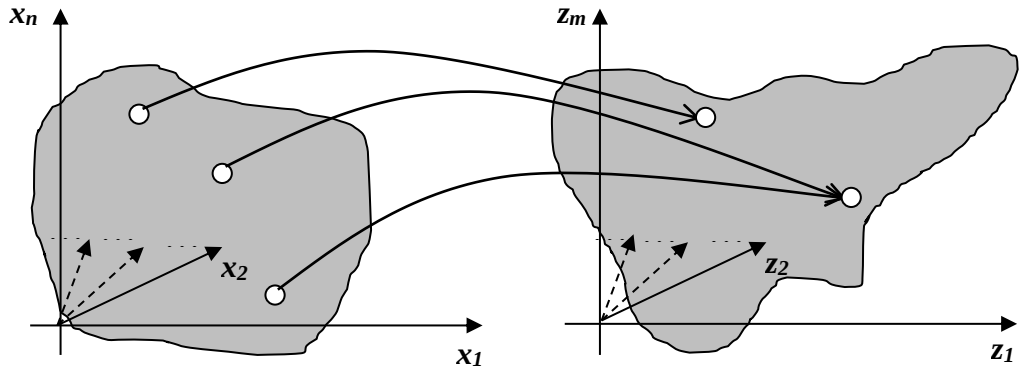


Figure 2.2. Decision space vs. objective space

Figure 2.2 depicts the decision variable space versus objective space. The Pareto optimal solutions for minimization problem are illustrated in Figure 2.3 for two objective functions. Since the problem illustrated in this figure is the minimization problem, the points in bold are the solutions belonging to the Pareto optimal set, such that they are the best solutions, since they dominate all other solutions, which are shown in circle.

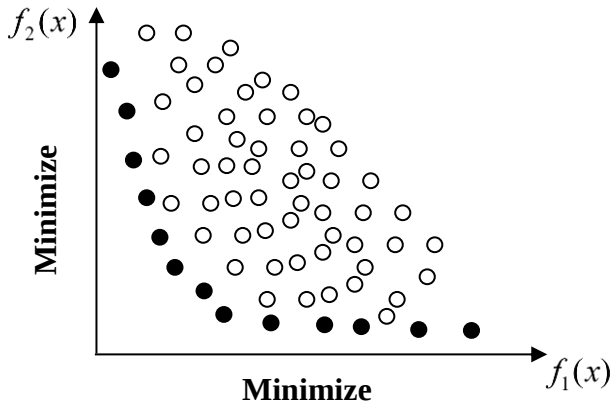


Figure 2.3. Pareto optimal solutions

Chapter 3

IMPERIALISTIC COMPETITIVE ALGORITHM AND ITS APPLICATIONS

3.1 Literature Review

Imperialist competitive algorithm (ICA) is based on inspirations from imperialistic competition. Accordingly, based on the idea of imperialism, the two fundamental components of ICA are empires and the colonies. Initially, the algorithm builds several randomly initialized empires where each empire includes one emperor and several colonies. Competitions take place between the empires and these competitions result in development of more powerful empires and the collapse of the weaker ones [2]. ICA is a population based metaheuristic that is inspired from observations on imperialists and their colonies. In this respect, there are other well-known metaheuristics inspired from biological and natural phenomena. Among these algorithms Genetic Algorithms (GAs) that conducts search and optimization procedures by imitating the process of natural evolution [3]. Other examples of such algorithms are Ant Colony Optimization (ACO), which is inspired by the behavior of ants in nature while looking for forage [4], and Particle Swarm Optimization (PSO), which is based on the social behavior of bird flocks in traveling long distance with guidance of local and global group leaders [5].

PSO mimics the behavior of a set of social insects that work together while exploring the search space. On the other hand, though ICAMA is similar to PSO in a way that

it also has a leader – imperialist, the behavior of colonies are quite different, since all of them try to move towards the best position of an imperialist without taking into consideration local best positions unlike in PSO. Differential Evolution (DE) [35] is another population-based method for optimizing a problem, which is very simple, but very powerful. DE is a deviant variety of GA, which iteratively tries to improve solutions with regard to a special type of differential operator. Evolutionary Strategy (ES) [36] also belongs to a class of evolutionary algorithms and it is inspired by the principles of biological evolution. ES algorithm solves optimization problems in an iterative manner by generating new offspring in every generation from the current population (parents) and selects the best ones for the next generation as the current population. Artificial Bee Colony (ABC) [34] is an algorithm inspired by the behavior of honey bees, in which individuals (solutions) are modified by the artificial bees, where the aim of bees is to discover the individual with the highest nectar. The idea of having two types of artificial bees – employed and onlooker bees allows algorithm to combine local search with global search methods, which is implemented by employed and onlooker bees respectively. Mean-Variance Mapping Optimization (MVMO) [68] is from among the recent algorithms that were developed in the field of heuristic optimization. The main idea behind MVMO is the approach of using a single parent-offspring pair and a normalized range of the search space, which is used for optimization variables. Moreover, the use of a special mapping function that is responsible for the actual mean and variance of the normalized optimization variables for mutation operation is another feature of MVMO. An improved version of Self Regulating Particle Swarm Optimization (iSRPSO) [69] is an algorithm that uses the method in which least performing particles that have different perceptions adopt different strategies for updating mechanism. The usage of these particles, the

best and the top three best particles, allows algorithm to find a good directional update for better solutions. HumanCog [70] is a 3-layer architecture algorithm for solving optimization problems that imitates a human behavior in terms of thinking and decision making, such that human cognitive and metacognitive behavior. Therefore, three layers are formed by cognitive, metacognitive and social cognitive layers. Thus, an optimal and accurate decision is made with the help of interaction of these three layers. CMA-ES_QR is another algorithm for solving single objective optimization problems, which is a variant of CMA-ES [71] for expensive scenarios. In addition to CMA-ES_QR algorithm, tuned CMA-ES (TunedCMAES) [72] is also a variant of CMA-ES for solving expensive problems, which uses bi-level optimization approach for tuning parameters of CMA-ES algorithm.

A particular example of trajectory based metaheuristics is the Simulated Annealing (SA) algorithm that is inspired from physical annealing process of metals. SA is currently the only metaheuristic for which a mathematical proof of finding a globally optimal solution exists under asymptotic computational conditions [6]. In fact, Reeves et al. introduced a convergence proof for GAs where the evolutionary procedure is modeled using Markov chains [38]. The authors considered the case of (1+1)-GAs where a 1-individual population generates one offspring per generation. Even though the proof is a pioneering step towards the convergence proof of GAs, it has limited contribution to practical use of evolutionary algorithms.

As a population based metaheuristic, ICA is applied for the solution of a wide variety of optimization problems mostly from engineering domains. In this respect, many problems that are already solved by evolutionary and nature-inspired metaheuristics are re-solved using ICA and comparative performance evaluations are carried out

and published in literature. ICA is successfully applied for the solution of assembly line balancing [7, 8], facility layout problems [9, 10], network flow problems [11, 12], supply chain management [13, 14], image processing [15, 16], artificial neural network training [17, 18], data mining [19, 20], power system optimization [21, 22], and scheduling [23, 24].

Since its initial introduction, several proposals on the improvements of ICA have been published in literature. These improvement proposals are either on strategical changes in algorithm's parameters and/or procedures, or on hybridization of ICA with other well-known soft computing methods. There are four fundamental parameters in ICA, these are namely assimilation rate (AR) representing the percentage of similarity between an imperialist and its colonies, revolution rate (RR) representing the replacement rate of weakest colonies by newly generated countries, ξ that weights the mean power of colonies on the total power of their empire and the pair ($N_{country}$,N_{empire}) describing the total number of colonies (countries) and number of empires. So far, studies on application of ICA for different problems proposed experimental tuning of these parameters and a detailed study describing the effect of parameter settings on the success of ICA algorithm is not found in literature. Bagher M, Zandieh M, Farsijani H have studied the effects of three classes of parameter settings for an assembly line problem and, over 9 degrees of freedom of the four algorithm parameters, their optimal values are obtained using Taguchi experiment design framework [25].

Considering the hybridization efforts of ICA with other soft computing methods, Talatahari S, Azar B F, Sheikholeslami R, Gandomi A H introduced chaotic improved ICA (CICA) where the authors studied seven different chaotic maps to

improve the assimilation procedure of conventional ICA and they reported that the best performance of CICA is obtained with the use of logistic and sinusoidal maps [25]. T. Niknam, E. Taherian Fard, N. Pourjafarian, A. Rousta proposed an efficient hybrid algorithm based on a modified ICA and K-means method and called the new algorithm as K-MICA. This hybrid method is used for finding the optimum clustering of N objects into K clusters. K-MICA was tested for robustness and ability of overcoming locally optimal solutions. Based on the comparative evaluations against several metaheuristics such as ant colony optimization (ACO), particle swarm optimization (PSO), genetic algorithms (GA), tabu search (TS), honey bee mating optimization (HBMO) and K-means, the obtained results exhibited that K-MICA performed better than its well-known competitors [27]. N. Razmjooy, B.S. Mousavi, F. Soleymani proposed a new hybrid algorithm that combines ICA and Neural Network (ICA-ANN) to solve skin classification problems. The authors used a multi-layer perceptron network (MLP) as a pixel classifier whereas an ICA was used for optimizing the weights of MLP [28]. Nia A R, Far M H, Niaki S T A built a hybrid of genetic algorithms and ICA for the solution of nonlinear integer programming problems [29]. In their proposed algorithm (HGA), ICA is first used to produce the best initial solutions for GAs and then GAS runs until a termination condition to improve the individuals in the initial population. Over six numerical examples in three categories (small, medium and large size), the experimental results showed that the proposed hybrid procedure could find better and nearer optimal solutions compared to those found by ICA and GAs.

As briefly mentioned above, the two fundamental components of ICA are the empires and the colonies, and the imperialistic competition is the most important phase of the algorithm. This competition causes the colonies to converge towards

locally optimal solutions within the search space with guidance of their corresponding imperialists. Hence, it is seen that strategies for moving the colonies towards their relevant imperialist – *assimilation* and generation of new countries in each empire, are the key procedures for the success of the algorithm.

3.2 Review of ICA

In imperialist competitive algorithm, the empires from among all countries (imperialists) compete for gaining the colonies, as the aim of each empire is to possess more colonies. This competition along with *assimilation* – moving colonies towards their relevant imperialists and *revolution* – changing the socio-political characteristics of colonies, enables the algorithm to search for a locally optimal solution that may also probably be the global optimum of the underlying objective function. During the competition among the empires, there is possibility that some colony will become better than the imperialist that it belongs to. In such a case, ICA swaps the positions of imperialist and this colony, so that the colony becomes the imperialist and the former imperialist becomes a colony. For a minimization problem, the power of an empire is inversely proportional to its fitness value. That is, the less is the fitness of the empire, the more powerful it is [2]. When there are no countries in an empire, it is termed to be *powerless* and the powerless empires collapse and terminate. In other words, when an empire runs out of the colonies and it has no more colonies in its state, then this empire becomes powerless and is eliminated by ICA. Thus, the number of empires decreases by one and this process continues until there remains only one the most powerful imperialist state. At this point, the algorithm may stop, since only one empire remained or it can continue till the maximum number of iterations specified by the user is reached. In practical implementations, it is recommended to continue till the satisfaction of termination

condition(s), since there is no guarantee that when only one empire remains the optimum solution is found.

3.2.1 Assimilation

Assimilation is the movement of colonies towards imperialist in their empire. This process is significantly effective on the success of ICA, as it is concerned with the improvement of colonies within the empires. Figure 3.1 describes the movement of a colony towards its associated imperialist in a randomly deviated direction to search the space around the imperialist. As shown in Figure 3.1, assuming that the dimension of the optimization problem is two, the current and the updated positions of a colony are denoted with a white and a black circle, respectively. Considering that the position of imperialist is (x_i,y_i) and the position of the colony is (x_c,y_c), the distance vector is $D=(x_i-x_c,y_i-y_c)$. A uniformly distributed and scaled random vector d is generated and added to current position of the colony to compute its new position. In Figure 3.1, the parameter Θ is also a random variable with uniform distribution between $(-\gamma, \gamma)$, where γ is a parameter that is used for adjusting the change in movement of a colony from the original direction [2].

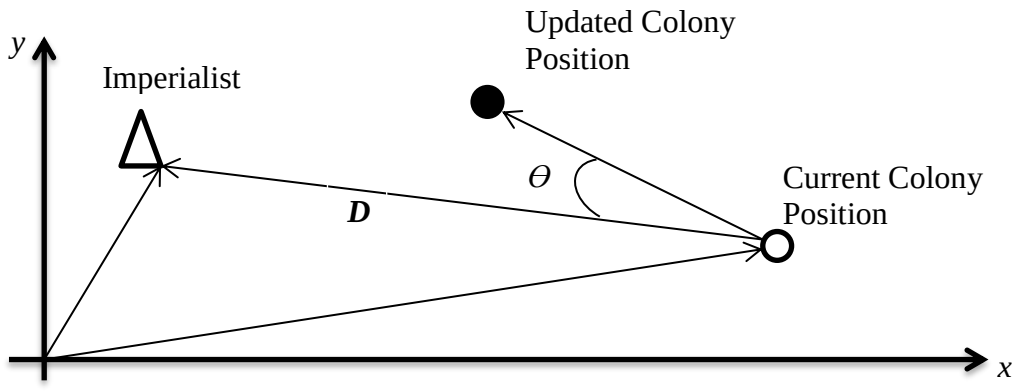


Figure 3.1. Moving a colony towards its relevant imperialist in a randomly deviated direction [2]

The assimilation procedure generalized to *n* dimensional problems is as follows:

Let

$$\boldsymbol{Col\_Pos} = [p_1, p_2, \ldots, p_n] \tag{3.1}$$

be the vector representing the colony's position and

$$\boldsymbol{Imp\_Pos} = [p'_1, p'_2, \ldots, p'_n] \tag{3.2}$$

be the vector representing its imperialist's position, where *n* is the dimension of the optimization problem. Now, let *D* be the vector containing the element-wise difference of (3.1) and (3.2),

$$\boldsymbol{D} = [p_1 - p'_1, p_2 - p'_2, \ldots, p_n - p'_n]. \tag{3.3}$$

Obviously, $\boldsymbol{D}$ is the vector representing the positional difference (*Col_Pos – Imp_Pos)*. Consequently, we proceed with the calculation of the new colony's position as,

$$\boldsymbol{Col\_Pos\_New} = \boldsymbol{Col\_Pos} + \beta \times \vec{\boldsymbol{r}} \otimes \vec{\boldsymbol{D}}, \tag{3.4}$$

where $\vec{r}$ is a uniformly distributed random variable vector of length *n*. β is a fixed algorithmic parameter that is commonly chosen to be about two [2].

3.2.2 Revolution

Revolution is the process of generating new countries within an empire. This happens by random changes in positions of some colonies [2]. Revolution is similar to the mutation operation in GAs where values of some variables are changed by randomly selected values with a very small probability. As a result, while new countries (colonies) are generated through revolution, some of the old colonies are replaced by the newly created countries. Pseudo code of the revolution procedure is presented in Algorithm 3.1.

Algorithm 3.1. Revolution

1. **For** each *empire* **do**
2. *Generate (RevolutionRate × NumberOfColoniesInEmpire) number of random countries.*
3. *Replace existing countries by newly created ones randomly.*
4. **EndFor**

3.2.3 Imperialistic Competition

Imperialistic competition takes place after assimilation and revolution operations. To describe the details of imperialistic competition, we need to discuss the computation of the total cost of an empire. The total cost of an empire can be expressed as follows [2]:

$$TC_k = IC_k + \varepsilon \times mean(CE_k), \tag{3.5}$$

where TC_k is the total cost of an empire k, IC_k is the imperialist cost of empire k, CE_k is the cost of the colonies of the empire k and ε is a constant parameter. A small value of ε causes the total cost of an empire to depend mostly on the imperialist, whereas a greater value for ε will make the total cost depending on both the imperialist and the colonies of the empire.

Competition among the empires is realized by removing the weakest empire from the competition and allowing other empires to compete between each other for the weakest colony in the weakest empire, which is excluded from the competition. Therefore, the following mathematical formulation describes the possession probabilities of the competing empires for the weakest colony [2].

$$p_k = \left| \frac{NTC_k}{\sum_{i=1}^{N} NTC_i} \right|, \tag{3.6}$$

where p_k is the possession probability of empire *k*, *N* is the number of imperialists and NTC_k is a normalized total cost, which is computed as follows:

$$NTC_k = TC_k + max(TC_i), i=1...N. \tag{3.7}$$

The final step in the competition between imperialists is to have a vector containing differences between possession probabilities and the uniformly distributed random values between (0, 1) as follows:

$$D = [p_1 - r_1, p_2 - r_2, ..., p_N - r_N], \tag{3.8}$$

where *N* is the number of imperialists.

The possessor of the weakest colony in the weakest empire is the one whose corresponding index in the vector *D* contains the maximum value. A detailed algorithmic description of ICA is presented in Algorithm 3.2 below.

Algorithm 3.2. ICA Algorithm

1. Initialize the population and create empires.
2. Compute the total cost of all empires.
3. **Do**
4. **For each** empire **do**
5. Apply *Assimilation* by moving colonies towards imperialist
6. Apply *Revolution* by replacing colonies based on revolution rate with newly created ones randomly.
7. Exchange position of an imperialist and the colony with the better cost if exists.

8. **EndFor**
9. Compute the total cost of all empires.
10. Apply imperialistic competition.
11. Eliminate empires which have no colonies.
12. **Until** termination condition is satisfied.

3.3 Application of Modified ICA for the Solution of Travelling Salesman Problem (TSP)

ICA is used for a solution to a well-known combinatorial problem named as travelling salesman problem. TSP is one of the most studied problems in optimization, which was first formulated in 1930. There are many different heuristics and methods proposed in the literature for solving TSP, which is NP-hard problem in combinatorial optimization. The idea behind TSP is simple, which can be stated as follows: there is a traveler, who wishes to visit *n* cities exactly once each by starting from a particular city and returning to it. Then objective is to find the shortest route for this traveler.

3.3.1 Formulation of TSP

TSP can be formulated as follows: let *n* be the total number of cities to be visited and $C = [c_{i,j}]$ be an $n \times n$ matrix containing costs (or distances) between cities, where $c_{i,j}$ denotes the cost of travelling from city *i* to city *j*. The objective is to find the shortest route among all given cities, where cost (or distance) matrix among all cities is given as input. The total cost *N* of a TSP tour for *n* cities is given by

$$N = \sum_{i=1}^{n-1} C_{i,i+1} + C_{n,1}$$

3.3.2 Modified ICA

Every colony in an empire changes its position in two parts of the algorithm, which are assimilation and revolution as was mentioned above. ICA in its original form is very powerful for solving real valued functions, where assimilation and revolution are very suitable. However, for solving TSP these two parts – assimilation and revolution need to be changed so that the algorithm can be applied to TSP. Therefore, in assimilation part *2-opt* local search is introduced together with the method of swapping the cities of the countries. The pseudo code for assimilation is given in Algorithm 3.3.

Algorithm 3.3. Assimilation(current_colony)

1. **Repeat** until there is no improvement
2. Best_distance = calculate_total_distance(current_route)
3. **For** all cities i **do**
4. **For** all cities k **do**
5. New_colony = 2optswap(current_colony, i, k)
6. New_distance = calculate_total_distance(New_colony)
7. **If** New_distance < Best_distance **Then**
8. current_colony = New_colony and **go to** 2
9. **Otherwise go to** 4
10. **EndFor**
11. **EndFor**
12. **EndRepeat**

Where calculation of total distance is done by using either Euclidean distance, geographical distance etc., depending on the type of the edge weights of the dataset used. On the other hand, 2opt swap is done by reversing the cities in a solution (colony) between points *i* and *k*, and leaving the rest unchanged.

Revolution part of the algorithm is changed to implement fragmentation method [30] on the countries and searching for the shortest route. This method of fragmentation is implemented as follows. Firstly, the candidate solution is divided randomly into several fragments based on the number of cities. Then construction of a new solution starts by choosing randomly first fragment. After that, the distances $\mathbf{d_n}$ between the last city in the first fragment and first cities of all other fragments are calculated. Additionally, the distances $\mathbf{d}'_n$ between the last city in the first fragment and the last cities of all other fragments are calculated. The fragment whose city is the closest to the last city in the first fragment is concatenated as it is if its first city is the closest one, otherwise if it is the last city which is the closest one, then the fragment is reversed before concatenation. This process lasts until all fragments are reconnected with each other. Moreover, this process in revolution part is applied to all the colonies of an empire. Figure 3.2 demonstrates an example of the fragmentation method used for revolution process. Algorithm 3.4 demonstrates the complete algorithm of ICA for TSP.

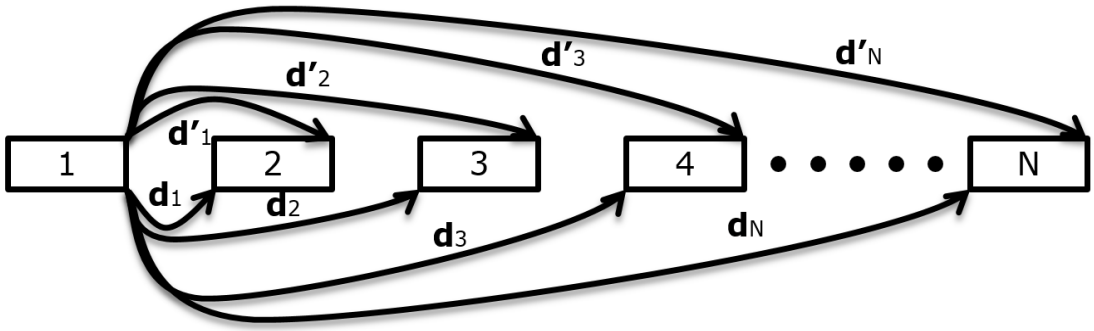


Figure 3.2. Demonstration of the fragmentation method for revolution process

Algorithm 3.4. ICA Algorithm for TSP

1. Initialize the population and create empires.
2. Compute the total cost of all empires.

3. **Do**
4. **For each** empire ***e*** **do**
5. **For each** colony ***c*** in empire ***e*** **do**
6. Assimilation(***c***)
7. Revolution(***c***)
8. Exchange position of an imperialist and the colony with the better cost if exists.
9. **EndFor**
10. **EndFor**
11. Compute the total cost of all empires.
12. Apply imperialistic competition.
13. Eliminate empires that have no colonies.
14. **Until** termination condition is satisfied.

3.4 Experimental Results

In order to test the modified ICA for solving TSP many sample instances from TSPLIB library were used. The proposed solution in this study for TSP provides good results for many sample instances. In all experiments the population is set to 200, where initial number of imperialists is set to be 8. Maximum number of iterations is 1000. The following Figure 3.3 illustrates the result for the berlin52.tsp instance sample from TSPLIB.

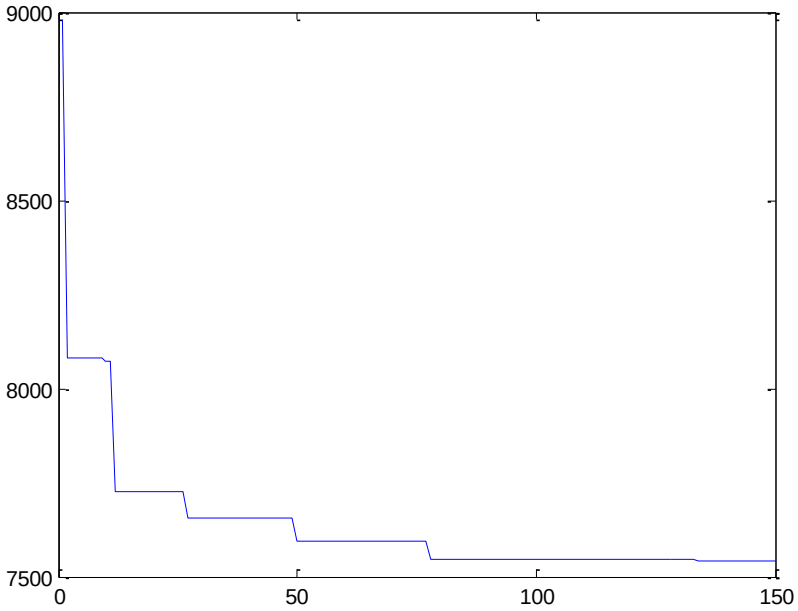


Figure 3.3. Obtained optimal result for berlin52.tsp instance with cost = 7542

The above figure shows how fast ICA converges to the optimal solution with cost equal to 7542, which is obtained in 134 iterations. The next Figure 3.4 demonstrates result obtained for eil101.tsp.

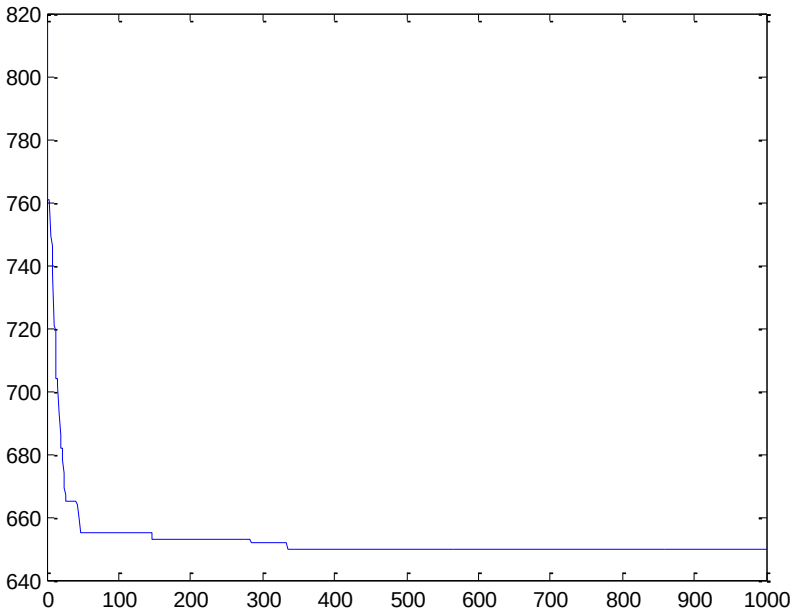


Figure 3.4. Obtained result for eil101.tsp with cost = 650

As shown in above figure, ICA again very quickly converges to the optimal solution. So, in the first 400 iterations it finds the path with cost equal to 650, which is very close to the optimal cost that is equal to 649. The following Table 3.1 summarizes some of the experimental results obtained in this study. The following sample instances from TSPLIB are used for experiments: berlin52.tsp, eil51.tsp, eil76.tsp, eil101.tsp, kroA100.tsp, kroC100.tsp, kroA150.tsp, kroB100.tsp, d198.tsp and rl493.tsp.

Table 3.1. Obtained best and average results of ICA for different instances

Problem	**Optimal**	**ICA Best Found**	**ICA Average**
berlin52.tsp	7542	7542	7542
eil51.tsp	426	427	427.5
eil76.tsp	538	546	547.5
eil101.tsp	649	650	650
kroA100.tsp	21282	21375	21378
kroC100.tsp	20749	20753	21322
kroA150.tsp	-	27567	27739
kroB100.tsp	-	22605	22942
d198.tsp	-	20208	20542
d493.tsp	-	41190	41698

In above table optimal values indicates with '-' are not found ones. As can be seen from the results obtained in this work, imperialist competitive algorithm along with 2-opt local search and fragmentation method produces good results. This is because of the search mechanism imperialistic competition in ICA, which enables it to search the whole search space.

Chapter 4

IMPERIALISTIC COMPETITIVE ALGORITHM WITH UPDATED ASSIMILATION FOR SOLVING SINGLE-OBJECTIVE OPTIMIZATION PROBLEMS

The assimilation procedure used in conventional ICA results in slow convergence speed in reaching the global optimum and sometimes it is the main cause of getting stuck at locally optimal solutions. In assimilation operation of ICA, even if there is a small deviation Θ in direction towards the imperialist, the direction still goes towards the imperialist since the current imperialist is the optimum solution of that empire. However, this might be misleading for the colonies due to the fact that imperialists are locally optimal solutions that may be far from the globally optimal position. Of course, after a number of iterations ICA may realize that there is a better position and replace the imperialist, as it does when a colony becomes better than the imperialist, but, as mentioned before, this slows down the performance and convergence to the global minimum of the objective function. Based on these observations, the proposed assimilation strategy aims to perform a better search around the imperialists and, compared to the original ICA proposal, both the convergence speed and the quality of the resulting solutions are improved significantly.

4.1 The Proposed Assimilation Strategy

The assimilation operation in ICA is modified in such a way that the colony is either moved towards the imperialist as in the original ICA or it is moved in a randomly selected direction scaled by the Euclidean distance between a colony and the its

imperialist. The selection between the two move operations is controlled by a parameter a_r that controls the percentage of assimilation operations of either type. The mathematical formulation of the modified assimilation operation is given below: Let *Col_Pos* be defined as in (3.1). then,

$$Col\_Pos\_New = \begin{cases} (Col\_Pos + \beta \times r \times ED) \times \theta, \ if \ \ rand() \le a_r \\ Col\_Pos + \beta \times r \times D \times \theta \quad otherwise \end{cases}, \tag{4.1}$$

where *ED* is the Euclidean distance between the colony and its imperialist, $\vec{r}$ is an n-dimensional random vector, $\vec{D}$ is the distance vector between the colony and its imperialist and the vector multiplication is performed element wise.

Another modification on the assimilation operation of original ICA is the computation of Pearson correlation coefficient (Pcc) between the colonies and their imperialists and using this value as an acceptance criterion for the new colony position. If Pcc is less than a predefined limit, ν, the new colony position is not accepted and the colony keeps staying in its current position. The Pcc is calculated as follows:

$$Pcc = \frac{n\sum x_i' y_i - \sum x_i' \sum y_i}{\sqrt{n\sum (x_i')^2 - (\sum x_i')^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}, \tag{4.2}$$

where x_i' *and* y_i, $i=1,..,n$, represent the assimilated position of the colony and position of the imperialist, respectively. The pseudo code of the proposed assimilation procedure is illustrated in Algorithm 4.1.

Algorithm 4.1. Modified Assimilation

1. **For** each *colony* in the empire **do**
2. Calculate Euclidean distance *ED* and element-wise
3. distance *D* between the *colony* and the *imperialist*

4. **If** rand() < a_r
5. $colony\_new = \{colony + \beta \times \vec{r} \times ED\} \times \Theta$
6. **Else**
7. $colony\_new = colony + \beta \times \vec{r} \times \vec{D} \times \Theta$
8. **EndIf**
9. Compute the cost of *colony_new*
10. **If** *colony_new_Cost* < *colony_Cost*
11. *colony = colony_new*
12. **Else**
13. Compute Pcc between *colony_new* and its *imperialist*
14. **If** |*Pcc*| >= ν **then**
15. *colony = colony_new*
16. **EndIf**
17. **EndIf**
18. **EndFor**

The flowchart of ICAMA algorithm is given in Figure 4.1.

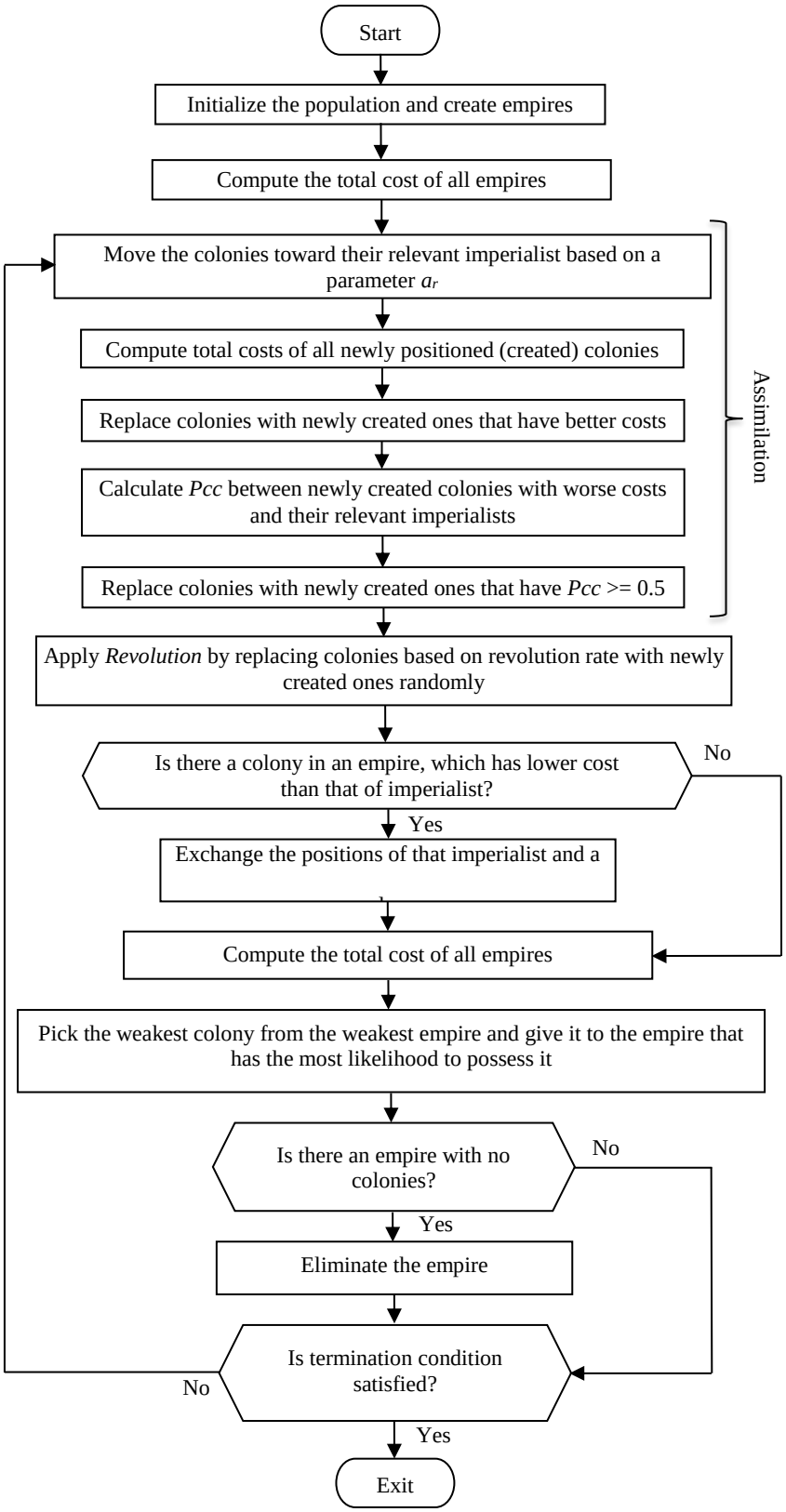


Figure 4.1. ICAMA algorithm flowchart

4.2 Experimental Results

Experimental evaluations are conducted using three sets of benchmark problems, namely the set of 23 classical benchmark problems [31], CEC2005 benchmarks [32] and CEC2015 benchmarks [33]. Results and discussions associated with each group benchmark problems are presented in subsections given below.

4.2.1 Experimental Evaluations with Classical Benchmark Problems

Problems in this set are divided into 3 groups – unimodal functions, multimodal functions with many local minima and multimodal functions with a few local minima. Unimodal functions are – *Sphere Model* (F_1), *Schwefel's Problem 2.22* (F_2), *Schwefel's Problem 1.2* (F_3), *Schwefel's Problem 2.21* (F_4), *Generalized Rosenbrock's Function* (F_5), *Step Function* (F_6) and *Quartic Function with Noise* (F_7). Multimodal functions with many local minima are – *Generalized Schwefel's Problem 2.26* (F_8), *Generalized Rastrigin's Function* (F_9), *Ackley's Function* (F_{10}), *Generalized Griewank Function* (F_{11}) and *Generalized Penalized Functions* (F_{12}, F_{13}). Multimodal functions with only a few local minima are – *Shekel's Foxholes Function* (F_{14}), *Kowalik's Function* (F_{15}), *Six-hump Camel-Back Function* (F_{16}), *Branin Function* (F_{17}), *Goldstein-Price Function* (F_{18}), *Hartman Functions* (F_{19}, F_{20}) and *Shekel Functions* (F_{21}, F_{22}, F_{23}). Comparative evaluations are carried out with results of original ICA[1], Particle Swarm Optimization (PSO) [5], Artificial Bee Colony (ABC) [34], Differential Evolution (DE) [35] and Evolutionary Strategy (ES) [36]. All experiments are conducted with a maximum of 500000 objective function evaluations for all algorithms in the experimental suit. Each test problem was solved 30 times and the best found objective function values are compared for each algorithm. Problem sizes and variable ranges are the same ones used in the study [32]. Initial numbers of empires and of colonies are set to 8 and 200, respectively, for

the original ICA and the proposed method. For the original ICA, $\beta=2$ and $\theta \in [-1,1]$ as they are also used.[1] For the proposed method, $\beta=0.2$, $\theta \in [-1,1]$, $a_r=0.9$ and $\nu=0.5$ for all benchmark problem sets and experimental trials. Tables containing results in bold indicate the best performed algorithms. Table 4.1 demonstrates the results of the proposed method ICAMA for the 23 classical benchmark problems described above.

As shown in Table 4.1, the proposed method found the optimal solutions for most of the benchmark problems in this set. For those problems for which the optimal solution could not be located exactly, the solution extracted by the proposed method is quite close to the optimal one. The only exception is problem F14 for which the extracted solution is from the optimal. This is the Kowalik's function that has a flat valley with sharply rising corners and most of the locally optimal solutions stay in the flat part of the fitness landscape. Experimental results indicate that our proposal that uses fixed parameters for all functions should be improved with adaptive parameters so that its step lengths can be adjusted dynamically for functions like F14.

Tables 4.2, 4.3 and 4.4 illustrate best results of the proposed method and best results of five well-known metaheuristics for comparative evaluations. As shown in Table 4.1, the proposed method extracted much better solutions than its competitors for all the unimodal functions. While the solutions extracted by the proposed method are very close to the optimal ones, those extracted by the competitors are far from optimal ones. Particularly, comparisons with the original ICA clearly demonstrates the improvements brought by the proposed method. Additionally, these results exhibit that the proposed method can locate unimodal optimality quite efficiently.

Table 4.3 shows the results of the proposed methods and its competitors for multimodal functions. It is seen that, except two benchmark problems, the proposed method is still the best performing algorithm. Optimal solutions are found for most problems. For functions F12 and F13, ABC algorithm is the best performing method, whereas results of proposed method are close to optimal solutions but left behind those extracted by ABC algorithm. Again, compared to the original ICA, the assimilation operation with the better movement of colonies towards their imperialist and the use of Pearson correlation coefficient resulted in significantly better solutions. The obtained results also exhibit that ICAMA is also a better alternative compared to its other well-known competitors.

Table 4.1. Experimental results for 23 classical benchmark problems

Function	***Optimal***	***Best***	***Worst***	***Std***	***Mean***
F1	0	0,0000E+00	0,0000E+00	0,0000E+00	0,0000E+00
F2	0	9,3113E-305	2,1011E-295	0,0000E+00	2,1070E-296
F3	0	0,0000E+00	0,0000E+00	0,0000E+00	0,0000E+00
F4	0	6,1008E-308	9,9206E-299	0,0000E+00	1,1921E-299
F5	0	8,7487E-10	2,8703E+01	9,0767E+00	2,8703E+00
F6	0	0,0000E+00	0,0000E+00	0,0000E+00	0,0000E+00
F7	0	7,5917E+00	8,7325E+00	3,6816E-01	8,1902E+00
F8	-12569.5	-1,2569E+04	-1,2569E+04	1,5312E-01	-1,2569E+04
F9	0	0,0000E+00	0,0000E+00	0,0000E+00	0,0000E+00
F10	0	0,0000E+00	0,0000E+00	0,0000E+00	0,0000E+00
F11	0	0,0000E+00	0,0000E+00	0,0000E+00	0,0000E+00
F12	0	5,5994E-10	1,8939E-07	6,8811E-08	5,7097E-08
F13	0	6,4605E-09	6,3687E-07	2,5787E-07	2,2001E-07
F14	1	2,9821E+00	4,8957E+00	6,5890E-01	3,4291E+00
F15	0.00031	5,0837E-04	1,2223E-03	2,2090E-04	9,4701E-04
F16	-1.03163	-1,0315E+00	-1,0294E+00	6,5656E-04	-1,0310E+00
F17	0.398	3,9803E-01	4,0526E-01	2,2883E-03	4,0025E-01
F18	3	3,0004E+00	3,1790E+00	6,1713E-02	3,0710E+00
F19	-3.86	-3,8613E+00	-3,8341E+00	8,8620E-03	-3,8531E+00
F20	-3.32	-3,1568E+00	-2,9044E+00	8,0241E-02	-3,0210E+00
F21	10	-1,0153E+01	-5,0552E+00	1,6121E+00	-9,6434E+00
F22	10	-1,0403E+01	-5,0877E+00	2,5675E+00	-8,8083E+00
F23	10	-1,0536E+01	-1,0536E+01	3,6829E-05	-1,0536E+01

Table 4.2. Best found results for unimodal functions

FUN	*ICAMA*	*ICA*	*PSO*	*DE*	*ES*	*ABC*
F1	**0.0000E+00**	4.0884E-03	4.3785E-07	3.4719E-29	7.4938E-04	4.3980E-16
F2	**9.3113E-305**	3.7061E-01	2.2529E-03	5.4824E-17	8.2310E-03	1.2819E-15
F3	**0.0000E+00**	3.9780E+01	2.9244E-03	5.9049E-27	1.7850E-01	3.1536E+03
F4	**6.1008E-308**	4.3548E+00	7.7350E-04	1.9265E-03	9.4000E-02	1.8914E+01
F5	**8.7487E-10**	2.8371E+01	1.1978E-02	1.8633E+01	1.1842E+01	2.7685E-02
F6	**0.0000E+00**	4.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F7	**7.5917E+00**	7.9391E+00	1.0086E+01	8.5152E+00	8.0483E+00	1.1992E+01

Table 4.3. Best found results for multimodal functions

FUN	*ICAMA*	*ICA*	*PSO*	*DE*	*ES*	*ABC*
F8	**-1.2569E+04**	-5.6948E+03	-1.1502E+04	-8.5087E+03	-1.1859E+04	**-1.2569E+04**
F9	**0.0000E+00**	4.7043E+01	5.4999E-01	9.4563E+01	4.5704E-04	**0.0000E+00**
F10	**0.0000E+00**	2.4069E-01	2.5947E-04	7.5495E-15	4.2756E-03	3.6415E-14
F11	**0.0000E+00**	1.6054E-01	1.5456E-05	0.0000E+00	1.6950E-03	1.1102E-16
F12	5.5994E-10	4.2445E-01	3.1216E-07	5.5085E-29	1.5746E-06	**4.5866E-16**
F13	6.4605E-09	6.6897E-01	1.3308E-05	1.3065E-28	2.3481E-05	**4.3431E-16**

Table 4.4 lists the results for multimodal functions with a few local minima. As mentioned above, the proposed method performed poorly for F14 with the above stated settings of parameter values. However, when the same function is solved by ICAMA using $\Theta \in (-\pi, \pi)$, optimal solution with fitness value 0.994 is found. This verifies our above mentioned conclusion that the proposed method needs improvement with adaptive parameter values to adjust the step sizes based on its journey over the function landscapes.

Other than F14, the proposed method extracted almost optimal solutions for all functions, while all competitors also extracted optimal solutions for all functions within this set.

In order to determine statistical similarity of ICAMA's results with those of its competitors Friedman's aligned ranks test is conducted [37]. This test also orders all

algorithms based on their statistical ranks, which makes it possible to compare all algorithms under consideration based on their achieved fitness values. Tables 4.5 and 4.6 show the Friedman's test scores for ICAMA and its five competitors for the 23 benchmark functions. From Table 4.5 it is clear that ICAMA has the smallest p-value that indicates the smallest statistical similarity to its competitors. The calculations of Friedman aligned ranks test statistic is based on the definition below [37].

$$F_{AR} = \frac{(\mathrm{k}-1)\left[\sum_{j=1}^{k} \hat{R}_j^2 - (kn^2/4)(kn+1)^2\right]}{\{[kn(kn+1)(2kn+1)]/6\} - (1/k)\sum_{i=1}^{n} \hat{R}_i^2}$$

Where $\hat{R}_i$ and $\hat{R}_j$ are the rank totals for problem *i* and algorithm *j* respectively. F_{AR} is compared for significance with a χ^2 distribution with $k-1$ degrees of freedom.

Table 4.4. Best found results for multimodal functions with a few local minima

FUN	***ICAMA***	***ICA***	***PSO***	***DE***	***ES***	***ABC***
F14	2.9821	**0.9980**	**0.9980**	**0.9980**	**0.9980**	**0.9980**
F15	5.0837E-04	4.1605E-4	**3.0750E-4**	**3.0750E-4**	**4.9245E-4**	**4.4511E-4**
F16	**-1.0315**	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**
F17	0.39803	**0.3978**	**0.3978**	**0.3978**	**0.3978**	**0.3978**
F18	**3.0004**	**3.0000**	**2.9999**	**2.9999**	**3.0000**	**3.0000**
F19	**-3.8613**	**-3.8627**	**-3.8627**	**-3.8627**	**-3.0897**	**-3.8627**
F20	-3.1568	**-3.3223**	**-3.3223**	**-3.3223**	**-3.3223**	**-3.3223**
F21	**-10.1532**	**-10.1532**	**-10.1531**	**-10.1531**	**-10.1531**	**-10.1531**
F22	**-10.4029**	**-10.4029**	**-10.4029**	**-10.4029**	**-10.4029**	**-10.4029**
F23	**-10.5363**	**-10.5364**	**-10.5364**	**-10.5364**	**-10.5364**	**-10.5364**

Table 4.5. Friedman aligned ranks

FUN.	***ICAMA***	***ICA***	***PSO***	***DE***	***ES***	***ABC***
F1	42	89	71	55	80	61
F2	54	102	107	59	86	64
F3	43	136	138	58	93	137
F4	53	119	90	82	92	129
F5	128	132	131	127	134	94
F6	47	122	48	45	46	44
F7	120	121	125	124	123	126
F8	2	6	4	5	3	1

Table 4.5 (continued)

F9	49	133	130	135	78	68
F10	50	111	79	65	85	67
F11	52	95	87	51	88	66
F12	69	108	74	56	70	60
F13	72	109	91	57	73	62
F14	112	110	106	104	105	103
F15	83	77	84	75	81	76
F16	41	39	40	37	38	36
F17	101	99	100	97	98	96
F18	118	116	117	114	115	113
F19	30	29	28	26	27	25
F20	35	31	34	32	33	63
F21	24	16	17	15	23	14
F22	21	13	18	12	20	11
F23	19	9	10	8	22	7
SUM	1365	1922	1729	1439	1613	1523
AVG	59.347	83.565	75.173	62.565	70.130	66.217

Table 4.6 shows the computed Friedman aligned ranks (FAR) and p-values for all algorithms under consideration. Small p-value indicates almost no statistical similarity among the algorithms while the rank of ICAMA shows that ICAMA is the best performing algorithm against its five competitors. This fact indicates that the proposed method is highly competitive for the solution of the first set of classical real-valued benchmark functions.

Table 4.6. Friedman aligned ranks statistics

Algorithm	Average F_{AR} values
ABC	66.217
DE	62.565
ES	70.130
ICA	83.565
ICAMA	59.347
PSO	75.173
F_{AR}	**26.417**
p-value	**0.000074**

4.2.2 Experimental Evaluations with CEC2015 Benchmark Problems

Experimental results conducted with expensive benchmark functions taken from CEC2015 competition are illustrated in Tables 4.7 and 4.8. The maximum number of function evaluations was set to 500 and 1500 for 10 and 30 dimensions respectively. ICAMA is executed over 20 consecutive runs under the same conditions stated in CEC2015 competition publications and the obtained mean fitness values are compared to those obtained by algorithms that are attendees of CEC2015 competition. Tables 4.7 and 4.8 list the best, worst, mean and standard deviation scores achieved by ICAMA for the 23 problems in this set. Tables 4.9-4.23 present the mean scores of ICAMA, PSO, ABC, DE, ES, ICA and CEC2015 competition attendees in order from best to worst. It is seen that, even though ICAMA is not the best performing algorithm for any of the 23 benchmark functions, whereas it performs better than several of the state-of-the-art modern algorithms.

Table 4.7. Best, worst, mean and standard deviation scores achieved by ICAMA for the 15 CEC2015 competition benchmark problems with dimension of 30

Best	Worst	Std	Mean
2,0107207E+10	4,2614082E+10	5,7413166E+09	3,3172626E+10
6,0577784E+04	1,6366152E+05	3,5954231E+04	1,0625753E+05
3,5275720E+01	4,1632740E+01	1,7975595E+00	3,8540919E+01
6,5606878E+03	7,8077183E+03	3,9980142E+02	7,1993253E+03
2,9602700E+00	4,8423400E+00	5,9373617E-01	3,7087860E+00
3,9524700E+00	5,0849700E+00	3,5367639E-01	4,4016265E+00
4,7463810E+01	9,3036060E+01	1,0667535E+01	6,5927901E+01
4,0546322E+05	3,5903182E+06	7,4105429E+05	1,3799516E+06
1,2863220E+01	1,4012610E+01	3,1568785E-01	1,3602056E+01
4,8649327E+06	4,4980467E+07	1,2561727E+07	2,3658854E+07
1,0018000E+02	2,2698420E+02	3,0291072E+01	1,6341136E+02
7,1572490E+02	1,8871388E+03	2,9030305E+02	1,3180690E+03
5,7692300E+02	1,0027360E+03	1,2827438E+02	7,5557168E+02
2,8962000E+02	4,1853080E+02	3,8434184E+01	3,4273762E+02
1,0339450E+03	1,5162465E+03	9,5851093E+01	1,3987657E+03

Table 4.8. Best, worst, mean and standard deviation scores achieved by ICAMA for the 15 CEC2015 competition benchmark problems with dimension of 10

Best	Worst	Std	Mean
7,2841919E+08	1,0194125E+10	2,3090413E+09	3,6442322E+09
1,9470573E+04	1,0777212E+05	1,9754904E+04	4,2469607E+04
8,2877800E+00	1,2350140E+01	1,1132128E+00	1,0423698E+01
1,4406986E+03	2,3908080E+03	2,5255540E+02	1,9024281E+03
1,1156600E+00	3,8752900E+00	6,6936223E-01	2,7620210E+00
2,0742200E+00	4,5068100E+00	5,8496740E-01	3,4395960E+00
7,2487500E+00	4,5858810E+01	9,0114946E+00	2,8344773E+01
2,6016230E+01	3,7742005E+04	8,2869329E+03	4,2449487E+03
3,2112300E+00	4,3102200E+00	2,4547422E-01	3,9810535E+00
1,4428110E+05	5,7808783E+06	1,2323825E+06	8,1080559E+05
7,4199000E+00	4,7970200E+01	9,9987624E+00	1,9494080E+01
1,3086210E+02	5,7267540E+02	9,9565777E+01	2,9342591E+02
3,4178120E+02	6,0071040E+02	6,9121768E+01	4,1784038E+02
1,9892920E+02	2,2619870E+02	6,5128629E+00	2,1326971E+02
3,0032550E+02	5,3108700E+02	6,3962660E+01	4,3781097E+02

Table 4.9. Mean results for function 1 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	Result for D=10	Algorithm	Result for D=30
MVMO	1.93E+02	MVMO	2.09E+03
TUNEDCMAES	1.17E+06	CMAS-ES_QR	8.50E+05
CMAS-ES_QR	4.43E+06	TUNEDCMAES	1.52E+06
ISRPSO	7.40E+06	ISRPSO	7.19E+08
PSO	2.88E+09	PSO	2.07E+10
HUMANCOG	3.27E+09	ICAMA	3.32E+10
ICAMA	3.64E+09	DE	3.74E+10
ICA	6.75E+09	ICA	4.47E+10
DE	7.21E+09	HUMANCOG	4.74E+10
ES	9.77E+09	ES	8.12E+10
ABC	1.00E+10	ABC	9.20E+10

Table 4.10. Mean results for function 2 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	Result for D=10	Algorithm	Result for D=30
MVMO	1.68E-02	MVMO	6.93E+03
CMAS-ES_QR	2.58E+04	ISRPSO	7.67E+04
ISRPSO	3.19E+04	CMAS-ES_QR	9.17E+04
ICAMA	4.25E+04	ICA	9.65E+04
TUNEDCMAES	4.78E+04	ICAMA	1.06E+05
HUMANCOG	7.80E+04	HUMANCOG	1.13E+05

Table 4.10 (continued)

DE	8.88E+04	ES	1.30E+05
ES	1.50E+05	TUNEDCMAES	1.44E+05
PSO	1.62E+05	DE	1.45E+05
ABC	1.92E+05	PSO	1.84E+05
ICA	3.37E+05	ABC	2.06E+05

Table 4.11. Mean results for function 3 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**
CMAS-ES_QR	2.79E+00	CMAS-ES_QR	1.15E+01
ISRPSO	6.60E+00	TUNEDCMAES	2.43E+01
TUNEDCMAES	7.62E+00	ISRPSO	2.57E+01
MVMO	9.40E+00	MVMO	3.79E+01
ICAMA	1.04E+01	PSO	3.74E+01
PSO	1.07E+01	ICAMA	3.85E+01
HUMANCOG	1.12E+01	HUMANCOG	4.13E+01
DE	1.19E+01	ICA	4.18E+01
ICA	1.20E+01	ES	4.33E+01
ABC	1.23E+01	DE	4.35E+01
ES	1.26E+01	ABC	4.62E+01

Table 4.12. Mean results for function 4 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**
MVMO	4.65E+02	MVMO	1.43E+03
ISRPSO	9.25E+02	ISRPSO	5.41E+03
TUNEDCMAES	1.34E+03	TUNEDCMAES	6.11E+03
CMAS-ES_QR	1.73E+03	CMAS-ES_QR	6.68E+03
ICAMA	1.90E+03	ICAMA	7.20E+03
DE	1.96E+03	ES	7.42E+03
PSO	1.96E+03	DE	7.65E+03
ICA	2.06E+03	PSO	7.92E+03
HUMANCOG	2.09E+03	HUMANCOG	7.99E+03
ES	2.09E+03	ICA	8.09E+03
ABC	2.20E+03	ABC	8.81E+03

Table 4.13. Mean results for function 5 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**
MVMO	1.13E+00	MVMO	1.68E+00
ISRPSO	2.46E+00	TUNEDCMAES	3.13E+00
ICAMA	2.76E+00	ES	3.21E+00

Table 4.13 (continued)

TUNEDCMAES	2.77E+00	ICAMA	3.71E+00
HUMANCOG	2.82E+00	ISRPSO	4.24E+00
ICA	2.82E+00	ICA	4.27E+00
DE	2.91E+00	HUMANCOG	4.39E+00
ES	2.97E+00	DE	4.44E+00
CMAS-ES_QR	3.20E+00	CMAS-ES_QR	4.55E+00
ABC	3.22E+00	ABC	5.19E+00
PSO	4.02E+00	PSO	5.79E+00

Table 4.14. Mean results for function 6 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**
MVMO	3.26E-01	MVMO	5.20E-01
CMAS-ES_QR	4.17E-01	ISRPSO	6.35E-01
ISRPSO	5.29E-01	TUNEDCMAES	7.16E-01
TUNEDCMAES	6.00E-01	CMAS-ES_QR	7.28E-01
PSO	2.90E+00	PSO	3.58E+00
ICAMA	3.44E+00	ICAMA	4.40E+00
HUMANCOG	3.63E+00	DE	4.89E+00
ICA	4.02E+00	HUMANCOG	5.03E+00
DE	4.76E+00	ICA	5.07E+00
ES	5.97E+00	ES	7.03E+00
ABC	6.06E+00	ABC	7.68E+00

Table 4.15. Mean results for function 7 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**
CMAS-ES_QR	5.52E-01	MVMO	4.39E-01
ISRPSO	5.71E-01	ISRPSO	5.68E-01
TUNEDCMAES	6.31E-01	TUNEDCMAES	7.28E-01
MVMO	6.37E-01	CMAS-ES_QR	7.47E-01
HUMANCOG	2.74E+01	PSO	4.93E+01
PSO	2.32E+01	ICAMA	6.59E+01
ICAMA	2.83E+01	HUMANCOG	8.86E+01
ICA	4.26E+01	ICA	8.86E+01
DE	5.19E+01	DE	1.02E+02
ES	7.09E+01	ES	1.77E+02
ABC	7.11E+01	ABC	2.07E+02

Table 4.16. Mean results for function 8 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**

Table 4.16 (continued)

CMAS-ES_QR	4.68E+00	CMAS-ES_QR	1.74E+01
ISRPSO	5.03E+00	TUNEDCMAES	2.84E+01
TUNEDCMAES	3.68E+01	MVMO	4.03E+02
MVMO	4.14E+01	ISRPSO	6.26E+02
PSO	1.34E+03	PSO	9.56E+05
ICAMA	4.24E+03	ICAMA	1.38E+06
ICA	6.53E+03	ICA	4.09E+06
HUMANCOG	7.77E+03	HUMANCOG	5.24E+06
DE	2.21E+04	ABC	1.04E+08
ABC	4.50E+04	DE	1.71E+07
ES	6.56E+04	ES	5.75E+07

Table 4.17. Mean results for function 9 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**
ISRPSO	3.95E+00	MVMO	1.34E+01
CMAS-ES_QR	3.96E+00	CMAS-ES_QR	1.34E+01
ICAMA	3.98E+00	ISRPSO	1.36E+01
MVMO	4.01E+00	ICAMA	1.36E+01
ICA	4.08E+00	ICA	1.37E+01
HUMANCOG	4.16E+00	ES	1.38E+01
TUNEDCMAES	4.17E+00	HUMANCOG	1.39E+01
PSO	4.19E+00	TUNEDCMAES	1.39E+01
DE	4.20E+00	DE	1.40E+01
ES	4.25E+00	ABC	1.41E+01
ABC	4.25E+00	PSO	1.41E+01

Table 4.18. Mean results for function 10 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	**Result for D=10**	**Algorithm**	**Result for D=30**
MVMO	4.97E+02	MVMO	9.29E+04
CMAS-ES_QR	2.25E+05	CMAS-ES_QR	3.25E+06
ISRPSO	3.53E+05	TUNEDCMAES	4.89E+06
TUNEDCMAES	5.38E+05	ISRPSO	6.83E+06
ICAMA	8.11E+05	PSO	2.05E+07
HUMANCOG	1.19E+06	ICAMA	2.37E+07
PSO	1.46E+06	HUMANCOG	5.60E+07
ICA	2.05E+06	ICA	6.66E+07
DE	2.34E+06	DE	7.62E+07
ES	2.56E+06	ES	1.13E+08
ABC	2.92E+06	ABC	1.85E+08

Table 4.19. Mean results for function 11 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	Result for D=10	Algorithm	Result for D=30
ISRPSO	7.26E+00	TUNEDCMAES	2.11E+01
TUNEDCMAES	7.45E+00	CMAS-ES_QR	2.46E+01
CMAS-ES_QR	7.63E+00	ISRPSO	5.09E+01
MVMO	1.17E+01	MVMO	1.43E+02
PSO	1.68E+01	PSO	1.50E+02
ICAMA	1.95E+01	ICAMA	1.63E+02
HUMANCOG	2.16E+01	ICA	2.41E+02
DE	2.60E+01	HUMANCOG	2.76E+02
ICA	3.07E+01	DE	3.23E+02
ES	3.83E+01	ES	5.37E+02
ABC	4.44E+01	ABC	7.20E+02

Table 4.20. Mean results for function 12 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	Result for D=10	Algorithm	Result for D=30
ISRPSO	1.82E+02	CMAS-ES_QR	6.27E+02
MVMO	2.00E+02	ISRPSO	7.36E+02
CMAS-ES_QR	2.35E+02	TUNEDCMAES	7.66E+02
TUNEDCMAES	2.39E+02	MVMO	8.60E+02
ICAMA	2.93E+02	ICAMA	1.32E+03
HUMANCOG	3.08E+02	PSO	1.52E+03
ICA	3.57E+02	HUMANCOG	1.60E+03
ES	3.97E+02	ICA	2.05E+03
PSO	4.17E+02	DE	2.58E+03
DE	4.25E+02	ES	5.57E+03
ABC	4.51E+02	ABC	4.73E+04

Table 4.21. Mean results for function 13 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	Result for D=10	Algorithm	Result for D=30
MVMO	3.16E+02	MVMO	3.44E+02
CMAS-ES_QR	3.26E+02	CMAS-ES_QR	3.80E+02
ISRPSO	3.31E+02	ISRPSO	4.00E+02
TUNEDCMAES	3.47E+02	TUNEDCMAES	4.15E+02
PSO	4.04E+02	PSO	6.52E+02
ICAMA	4.18E+02	ICAMA	7.56E+02
HUMANCOG	4.33E+02	HUMANCOG	8.35E+02
DE	4.67E+02	DE	8.87E+02
ICA	5.62E+02	ICA	9.60E+02
ABC	5.81E+02	ES	1.55E+03
ES	5.91E+02	ABC	1.95E+03

Table 4.22. Mean results for function 14 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	Result for D=10	Algorithm	Result for D=30
CMAS-ES_QR	1.97E+02	CMAS-ES_QR	2.35E+02
ISRPSO	2.01E+02	TUNEDCMAES	2.47E+02
TUNEDCMAES	2.05E+02	ISRPSO	2.65E+02
MVMO	2.06E+02	MVMO	2.76E+02
ICAMA	2.13E+02	PSO	3.19E+02
HUMANCOG	2.15E+02	ICAMA	3.43E+02
PSO	2.16E+02	HUMANCOG	3.94E+02
ICA	2.20E+02	ICA	4.00E+02
DE	2.23E+02	DE	4.43E+02
ES	2.28E+02	ES	4.93E+02
ABC	2.33E+02	ABC	6.45E+02

Table 4.23. Mean results for function 15 of CEC2015 competition from best to worst with dimension sizes of 10 and 30

Algorithm	Result for D=10	Algorithm	Result for D=30
ISRPSO	3.00E+02	CMAS-ES_QR	4.90E+02
CMAS-ES_QR	3.79E+02	TUNEDCMAES	8.01E+02
PSO	4.18E+02	ISRPSO	9.51E+02
ICAMA	4.37E+02	MVMO	1.19E+03
TUNEDCMAES	4.42E+02	PSO	1.34E+03
HUMANCOG	4.74E+02	ICAMA	1.40E+03
MVMO	4.76E+02	DE	1.42E+03
ICA	4.82E+02	HUMANCOG	1.49E+03
DE	4.91E+02	ICA	1.59E+03
ABC	5.45E+02	ES	1.70E+03
ES	5.50E+02	ABC	1.83E+03

4.2.3 Experimental Analysis on the Strategy of Parameter *v*

An important parameter on the success of the proposed algorithm is the "v" parameter that determines the threshold of accepting an individual with fitness worse than its parent but it is highly correlated to its imperialist. Table 4.24 illustrates the results of ICAMA on CEC2015 benchmarks for five different values of *v*. These values are selected to exhibit the effect of *v* with values within the range (0, 1]. It can be seen from Table 4.24 that our setting v=0.5 results in the best performance.

Table 4.24. The results of ICAMA on CEC2015 problems for different values of ν

Problem	v=0.1	v=0.4	v=0.5	v=0.6	v=0.9
1	4.70E+10	4.87E+10	**3.32E+10**	4.85E+10	4.84E+10
2	9.30E+04	**8.54E+04**	1.06E+05	1.02E+05	1.14E+05
3	4.07E+01	4.04E+01	**3.85E+01**	4.04E+01	3.98E+01
4	7.85E+03	7.64E+03	**7.20E+03**	7.87E+03	7.71E+03
5	4.26E+00	4.30E+00	**3.71E+00**	4.33E+00	4.44E+00
6	5.22E+00	5.14E+00	**4.40E+00**	5.26E+00	5.13E+00
7	9.14E+01	9.52E+01	**6.59E+01**	8.81E+01	9.11E+01
8	5.03E+06	3.75E+06	**1.38E+06**	4.95E+06	4.26E+06
9	1.38E+01	1.38E+01	**1.36E+01**	1.37E+01	1.38E+01
10	5.51E+07	5.27E+07	**2.37E+07**	3.79E+07	4.12E+07
11	2.19E+02	2.19E+02	**1.63E+02**	2.35E+02	2.19E+02
12	1.69E+03	1.88E+03	**1.32E+03**	1.77E+03	1.62E+03
13	9.97E+02	8.46E+02	**7.56E+02**	9.32E+02	9.50E+02
14	3.99E+02	4.02E+02	**3.43E+02**	4.26E+02	3.76E+02
15	1.53E+03	1.52E+03	**1.40E+03**	1.50E+03	1.50E+03

Chapter 5

MULTI-OBJECTIVE IMPERIALISTIC COMPETITIVE ALGORITHM WITH MULTIPLE NON-DOMINATED SETS FOR THE SOLUTION OF GLOBAL OPTIMIZATION PROBLEMS

In this chapter, a multi-objective imperialistic competitive algorithm (MOICA) is discussed. MOICA is proposed for solving global multi-objective optimization problems. It is a modified and improved multi-objective version of a single objective ICA, which was previously proposed by Atashpaz-Gargari and Lucas [2]. The presented algorithm implements the idea of imperialism. Accordingly, individuals in a population are called countries, of which there are two types—colonies and imperialists. MOICA incorporates the competition between empires and their colonies for the solution of multi-objective problems. To this end, it employs a proposed approach of several non-dominated solution sets, whereby each set is called a local non-dominated solution set (LNDS). All imperialists in an empire are considered non-dominated solutions, whereas all colonies are considered dominated solutions. Aside from local non-dominated solution sets, there is one global non-dominated solution set (GNDS), which is created from LNDS sets of all empires. There are two main operators of the proposed algorithm: Assimilation and Revolution. They respectively use GNDS and LNDS sets during assimilation and revolution of colonies. The significance of this study is the notable feature of the

proposed algorithm; specifically, no special parameter is used for diversity preservation. This enables the algorithm to avoid extra computations in order to maintain the spread of solutions. Simulations and experimental results on the multi-objective benchmark problems showed that MOICA is more efficient compared to many existing multi-objective optimization algorithms because it produces better results for most of the test problems.

5.1 Literature Review

Many real-world problems must be solved by optimizing more than one objective. In some cases, one objective must be minimized while the other must be maximized [1]. Many multi-objective optimization algorithms have been proposed for optimizing several objectives. Among them are multi-objective evolutionary algorithms (MOEAs) [39, 40, 41, 42 and 43]. A priority of multi-objective optimization algorithms is to simultaneously find several Pareto-optimal solutions. Another priority is to additionally optimize conflicting objectives when one must be minimized and the other must be maximized. Consequently, multi-objective optimization algorithms have gained popularity in the last two decades. The aim of this study was therefore to develop a multi-objective optimization algorithm inspired by imperialistic competition—specifically, multi-objective imperialistic competitive algorithm (MOICA)—which uses a population of countries of two types: imperialists and colonies. In every empire, there is an imperialist, which is considered the local best for that empire. Accordingly, MOICA generates a local non-dominated set of solutions for each empire. It then finally calculates the global non-dominated set of local non-dominated solutions of each empire, which is the final set of non-dominated solutions.

Many applications of ICA exist, especially in the field of engineering. Only in the field of computer engineering ICA is applied to data clustering and image processing for solving such problems as skin color detection and template matching [44]. For example, Duan et al. [45] presented a template matching method based on chaotic ICA in which a correlation function is used. Those authors prevented the problem of falling into the local best solution by introducing a chaotic behavior of ICA, which improves the global convergence. Another example of the application of ICA is the integrated product mix-outsourcing optimization problem [46]. Vedadi et al. [47] applied ICA in the field of electrical engineering by presenting ICA-based Maximum Power Point Tracking algorithm to find Global Maximum Power Point of power-voltage string under Partial Shading Condition rapidly and precisely. Goudarzi et al. [48] used ICA as a heuristic technique for optimization procedure in finding the optimal location of capacitors in radial distribution systems. Another example of application of ICA is in the field of geoscience, where ICA is used for locating the critical failure surface and computing the factor of safety in a slope stability analysis based on the limit equilibrium approach [49]. Jordehi A R [50] proposed a solution to flexible AC transmission systems (FACTS) allocation problem in a way that low values of overloads and voltage deviations results both during line outage contingencies and demand growth. In this study, thyristor-controlled phase shifting transformers and thyristor-controlled series compensators have been used as FACTS devices. Besides applications of ICA, variants of ICA have been presented in the literature. For example, Ebrahimzadeh et al. [62] proposed a novel hybrid intelligent method (HIM) for recognition of the common types of control chart pattern (CCP). The proposed method includes two main modules: a clustering module and a classifier module. Authors used a combination of the modified imperialist

competitive algorithm (MICA) and the K-means algorithm in the clustering module for clustering input data. A mutation operator was also introduced into the proposed algorithm by changing assimilation process.

Aside from the variants of ICA and its applications there are many different algorithms in the literature for solving multi-objective optimization problems. Among them are the algorithms, which are used in experimental section of this chapter. For example, MOEAD (or MOEA/D) is a multi-objective evolutionary optimization algorithm, which is based on decomposition [74]. MOEAD decomposes a problem into several optimization sub problems using uniformly distributed aggregation weight vectors and performs optimization of these problems simultaneously, while every sub problem is optimized by use of the information of several surrounding sub problems. This makes MOEAD have a better computational complexity in every generation with comparison to many other state-of-the-art algorithms. Qi Y et al. proposed a variant of MOEA/D called MOEA/D-AWA, which is an improved MOEA/D with adaptive weight vector adjustment [60]. MOEA/D-AWA addresses situations when multi-objective optimization problem has a complex Pareto front, i.e. a discontinuous Pareto front or a Pareto front with sharp peak or low tail. Therefore, in this algorithm a new method for weight vector initialization along with an adaptive weight vector adjustment strategy are introduced. Additionally, MOEA/D-AWA has a feature of an external elite population, which enables new sub problems being added into scattered, i.e. discontinuous regions of the Pareto front. Harmony Search algorithm [89] is a single-objective optimization algorithm that was used by Doush I A and Bataineh M Q for hybridizing MOEA/D and NSGA-II [86] algorithms, thus obtaining Harmony MOEA/D and Harmony NSGA-II algorithms [61], which performed better than

original MOEA/D and NSGA-II algorithms. NSGA-II is a non-dominated sorting based MOEA, which is one of the most famous multi-objective optimization algorithms that addresses three problems for which MOEAs have been mainly criticized: 1) computational complexity; 2) non-elitism approach; and 3) use of sharing parameter. M. Reyes Sierra and C. A. Coello Coello proposed Optimized MOPSO (OMOPSO), which is a variant of multi-objective PSO algorithm [85]. OMOPSO uses the crowding distance of NSGA-II in order to eliminate some of the best solutions or so-called leader solutions. In addition to crowding distance, in order to accelerate the convergence of the swarm OMOPSO uses the combination of two mutation operators. Another feature of OMOPSO is that it uses a concept of ϱ-dominance in order to make the algorithm produce a limited number of solutions. Zitzler E, Laumanns M and Thiele L proposed SPEA2 [87], which is an improved version of its predecessor – Strength Pareto Evolutionary Algorithm (SPEA) [90]. Unlike SPEA, SPEA2 incorporates a fine-grained fitness assignment strategy along with a technique for density estimation and an improved method of archive truncation. DMCMOABC is a dynamic multi-colony multi-objective artificial bee colony algorithm proposed by Xiang Y and Zhou Y, which uses a strategy of dynamic information exchange and the multi-deme model [73]. In this algorithm several colonies mainly search individually and sometimes share information between each other. This algorithm involves employed and onlooker bees, which explore better positions in every generation. DMCMOABC makes use of an external archive for keeping best solutions, i.e. the non-dominated solutions, while diversity among archived solutions is preserved by using crowding distance. Authors of this algorithm used the *migration rate* parameter in order to replace the worst food source in a randomly selected colony by the elite intermediate individual with the maximum

crowding distance. S. Kukkonen and J. Lampinen proposed Generalized Differential Evolution 3 (GDE3), which is another algorithm with a diversity maintenance technique used in CEC2007 as well as in CEC2009 Special Sessions on Performance Assessment on Multi-Objective Optimization Algorithms [75]. GDE3 performed well against other state-of-the-art algorithms and it is one of the best selected algorithms in CEC2009. MOEADGM is an improved version of MOEAD with two mechanisms for better optimization, which is proposed by C.-M. Chen, Y.-P. Chen, and Q. F. Zhang [76]. First mechanism is the replacement of evolution operator with guided mutation operator for better utilization of information obtained from neighbors. Second mechanism is for the performance improvement, which is utilizing a priority queue for updating. L.-Y. Tseng and C. Chen proposed another multi-objective optimization algorithm called Multiple trajectory search (MTS), which was successfully applied to unconstrained and constrained set of problems in CEC2009 [77]. MTS algorithm first generates a set of uniformly distributed solutions, which are divided into two – foreground and background solutions. The search mechanism focuses mainly on the first type of solutions, that is foreground ones, while it focuses partly on others. The MTS selects one of the three local search methods, which it applies on solutions iteratively. In the beginning these methods search in a very large neighborhood, which decreases gradually until it becomes of pre-defined tiny size and resets to its initial size again. Utilization of such a size varying neighborhood searches enables MTS effectively solve optimization problems. H.-L. Liu and X. Q. Li proposed a multi-objective optimization algorithm called LiuLiAlgorithm [78]. This algorithm is based on sub-regional search by dividing the decision space into several lesser regions. The use of sub-regional search makes LiuLiAlgorithm perform better in terms of computational complexity. M. H.

Liu et al. proposed an improved version of DMOEA [91, 92] with domain decomposition technique DMOEA-DD [79]. This algorithm was one of the most successful competitors in CEC2009. K. Sindhya et al. proposed an augmented local search based evolutionary multi-objective optimization algorithm NSGAIILS, which is a hybrid evolutionary algorithm derived by a combination of NSGA-II and an augmented local search [80]. NSGAIILS was also tested on the set of unconstrained and constrained problems from CEC2009, where it could produce good, but not best results with comparison to other algorithms. V. L. Huang et al. proposed the multi-objective version of Self-adaptive Differential Evolution algorithm (SaDE) [93] with objective-wise learning strategies OWMOSaDE [65], which is an improved version of MOSaDE [88]. The original SaDE algorithm does not require pre-specifying of control parameters and the choice of learning strategy, since parameter settings and the learning strategy are self-adapted with the help of learning experience during evolution. MOSaDE was developed for the solution of numerical optimization problems with several conflicting objectives. OWMOSaDE learns mutation strategies and best values for crossover parameter specifically for every objective function. A clustering MOEA (ClusteringMOEA), which is based on orthogonal and uniform design, was proposed by Y. P. Wang et al. [81] in 2009. In order for ClusteringMOEA to estimate good points for more exploration during iterations it uses orthogonal design to create initial points in a population that are uniformly distributed over the solution space. A new crossover was designed for an efficient exploration of the search space and finding best solutions. The exploration in this algorithm mainly focuses on the boundary and sparse parts of non-dominated solutions, which are obtained in objective space. Finally, in order to improve the cardinality, i.e. the number of solutions finely distributed over the Pareto front, a

novel clustering approach was developed for selecting the non-dominated solutions. S. Tiwari et al. proposed a hybrid AMGA algorithm [82], which is a combination of a single-objective optimization with evolutionary multi-objective optimization algorithms. AMGA uses a classical gradient based algorithm as a single-objective optimizer for a fast local search. It behaves as a mutation operator, which is used as a genetic mutation as well as a gradient based mutation. On the other hand, AMGA uses multi-objective optimization algorithm as a global search. AMGA also uses a scalarization scheme for the improvement of objective functions, which uses reference points as constraints. This allows algorithm to solve non-convex optimization problems. B. Y. Qu and P. N. Suganthan proposed multi-objective evolutionary programming (MOEP) that uses fuzzy rank-sum with varied selection [65]. MOEP algorithm's performance is significantly faster than the performance of the same algorithm that uses non-domination sorting. A. Zamuda et al. proposed Differential Evolution with Self-adaptation and Local Search for Constrained Multi-objective Optimization algorithm (DECMOSA-SQP) [83]. This algorithm incorporates constrained handling mechanism along with a SQP local search and the self-adaptation mechanism taken from DEMOwSA algorithm [94]. S. Gao et al. proposed an orthogonal multi-objective evolutionary algorithm OMOEAII with lower dimensional crossover [84]. The lower-dimensional crossover enables algorithm to converge faster, since the search complexity is decreased. Moreover, by using orthogonal crossover the probability of finding better solutions increases.

5.2 Overview of MOICA

The proposed MOICA algorithm implements the idea of imperialism by incorporating the competition among empires. The main idea behind MOICA is that there are several non-dominated solution sets, i.e. imperialists, per each empire and

one global non-dominated solution set, which contains the best imperialists among all empires. All empires compete during the process and strive to take possession of the colonies of other empires based on their power. Therefore, all empires have the opportunity to assume control of one or more colonies of the weakest empire. During iterations of the algorithm, colonies of each empire make changes with respect to their positions as a result of changing their cost values. As previously mentioned, some colony ***C*** in an empire may become better than some of the current set of imperialists, say ***I***. In such a case, the new colony ***C*** with a better cost becomes a member of the empire's imperialists, that is, a member of the set of non-dominated solutions. Thus, previous imperialist ***I***, which is dominated by ***C,*** becomes a colony.

MOICA has an important yet simple feature in its implementation. Specifically, it has several non-dominated solution sets, which makes it different from many other multi-objective optimization algorithms. Initially, there is ***N*** number of empires. Therefore, every empire possesses Pareto optimal solutions, or local non-dominated solutions (LNDS). Therefore, the total number of LNDSs will initially be ***N***. Moreover, there is a set of global non-dominated solutions (GNDS), which is obtained from ***N*** number of LNDSs. Because the set of local non-dominated solutions for each empire is updated during iterations, the GNDS is also accordingly updated. This means that the algorithm has one GNDS throughout the implementation process, whereas the number of LNDSs will gradually decrease on account of the collapse of some empires during the competition. Figure 4.1 illustrates an example of three empires (E_1, E_2, and E_3) with their colonies and local non-dominated solution sets, i.e., imperialists, which are set in bold.

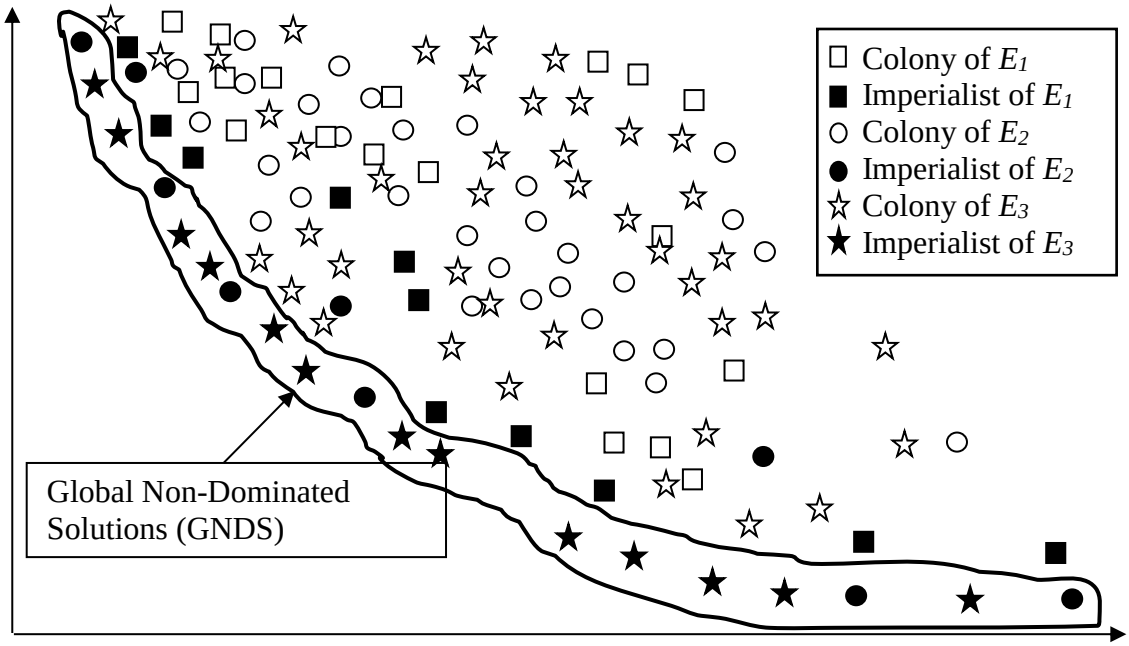


Figure 5.1. GNDS and LNDS sets of three empires

Imperialists that are taken into an area in Figure 5.1 are the best imperialists among all empires that form GNDS. There is a possibility that none of the imperialists will be included in GNDS of some empire. An example of such scenario is Empire 1 in Figure 5.1. Therefore, the use of GNDS in this algorithm is very important because colonies of all empires are assimilated toward the randomly selected imperialists from the GNDS, which enables an algorithm to avoid local optima. If we consider only one empire in Figure 5.1, for example, E_2, it is readily apparent that the circles in bold form non-dominated solution set empire E_2. The assimilation and revolution operations will be detailed in the following subsections.

Non-dominancy in the proposed algorithm is calculated based on fronts. Therefore, solutions that are assigned a value of 1 belong to the first front, while solutions with front value 2 are assigned to the second front, and so on. As a result, LNDS and GNDS sets contain solutions that belong to the first front only. Another significant feature in the proposed algorithm is that no special parameter is used for diversity

preservation, which enables the algorithm to avoid extra computations in order to maintain the spread of solutions. Although a share parameter is not used in MOICA, the spread of solutions in the results obtained from our simulations and experiments was very good. This was achieved on account of the assimilation technique used in this algorithm. That is, as described in Chapter 3, all colonies of an empire move toward one imperialist that is available in the empire. However, in the proposed algorithm, colonies of an empire move toward one of the imperialists, ***I,*** in the GNDS set. The imperialist ***I,*** toward which the colonies move, is randomly selected in each iteration from the set of global non-dominated solutions. Therefore, the idea of avoiding usage of a share parameter is derived from the nature of multi-objectiveness, in which every solution in a non-dominated solution set is considered a valid solution so that there is no single solution. For the sake of clarity, the description of the proposed algorithm is first provided in Algorithm 5.1. Then, each part of the algorithm is detailed.

Algorithm 5.1. MOICA Algorithm

1. Begin
2. Initialization:
    - Initialize problem parameters, such as objective function name, number of variables, and lower and upper bounds of decision variables.
    - Initialize algorithm parameters, such as population size, number of initial empires, number of iterations, and other coefficients used in assimilation and revolution operations.
    - Initialize population
3. Evaluate objective functions and assign objective values to each country.
4. Apply non-domination sorting [51].
5. Create initial empires by distributing colonies randomly, create LNDS for every empire and obtain GNDS.
6. **For each** iteration *i* **do**:

7. **For each** empire *j* **do**:
8. Apply *assimilation*: move colonies toward one of the randomly selected imperialists in the GNDS set and apply *economic changes* with probability p_e.
9. Apply *revolution*: Generate new countries from the LNDS set according to probability p_r and revolution rate α.
10. Evaluate objective functions and assign cost values to all colonies.
11. Update LNDS for empire *j* and update GNDS.
12. Calculate the total power of empire *j*.
13. **End for**
14. Unite similar empires.
15. Apply imperialistic competition and terminate powerless empires.
16. **End for**
17. Display results.

5.2.1 Non-Domination Sorting

Various methods have been proposed in the literature for non-dominancy. In these methods, each solution in the search space is assigned a rank value, which indicates whether a certain solution in the population is dominated by other solutions. In most cases, the lower the rank value is, the less this solution is dominated by others. For example, a rank value of one indicates that this solution is non-dominated. Another approach for non-dominancy is to not assign solutions a rank value; rather, divided them into fronts [51]. This is the approach used in this study for non-dominancy. Figure 5.2 illustrates non-dominated solutions with fronts for the minimization problem.

A front with a value of one contains non-dominated solutions, where the front value of two is the set of solutions dominated by the solutions from the first front only. Solutions with a front value of three are dominated by the solutions from the previous fronts. Therefore, in the proposed algorithm, every empire has its own local

non-dominated solutions, LNDS. This LNDS is therein intended to include imperialists of the empire; thus, there is no single imperialist in the empire. This means that all other solutions have front values greater than one, such that dominated solutions are considered colonies of the empire.

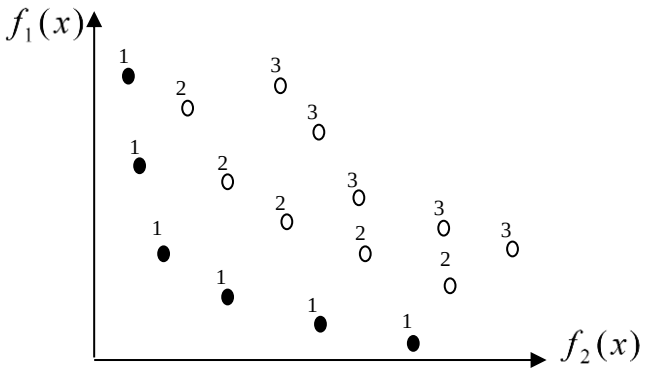


Figure 5.2. Non-dominancy using fronts

5.2.2 Assimilation

Assimilation, the movement of colonies toward imperialists, is implemented in a similar way as explained in Chapter 3. However, the difference is that there are several imperialists in the GNDS set. Thus, one of the imperialists should be selected and should serve as a target for the movement of colonies. The use of GNDS in assimilation instead of LNDS sets enables the algorithm to escape the local minima faster. The selection of the target imperialist is randomly performed for each empire in each iteration of the algorithm.

Figure 5.3 illustrates the assimilation implemented in this algorithm. In the figure, the black circles and red triangle indicate non-dominated solutions, the GNDS set, such that they are imperialists of the whole population. The red triangle is the randomly selected target imperialist toward which the colonies are moving. For

simplicity, only one moving colony is shown in the figure and is indicated by a blue circle. Parameters, such as θ, d and x are explained in Chapter 3. Therefore, their descriptions are omitted here because they have the same meanings. Nonetheless, the values used for some parameters are different, which will be discussed later. Another important point is that, owing to randomized selection of target imperialists and deviation θ, the diversity in the algorithm is preserved. In Figure 5.3 the angle is denoted with $\bar{\theta}$ because deviation θ is used in the decision space, which may not be the same as in the objective space. Therefore, even if deviations in decision and objective spaces differ, some deviation still exists in the objective space, which is denoted by $\bar{\theta}$.

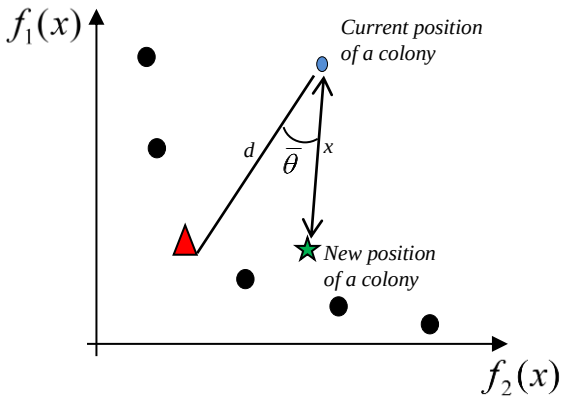


Figure 5.3. Assimilation of a colony towards randomly selected imperialist from the GNDS set

To improve the local search of the proposed algorithm, another new operation is added immediately after the assimilation process. This operation is the influence of *economic changes* on the empire, which has some probability of being engaged, as described in the pseudo-code below. The higher the value of p_e is, the lower the probability is for performing the operation. In most cases, the value of 0.9 is used for p_e to incite few economic changes. *UpperBound* and *LowerBound* are the vectors,

which indicate the decision space of the decision variables for the given objective function. *rand()* is a uniformly generated random value between (0, 1). The procedure of the assimilation process is given below. The variables and parameters, *Col_Pos_New*, *Col_Pos*, θ, β and *r* are the same as in Chapter 3. However, the variable, *d*, is different in this operation because it contains an element-wise difference of a colony and a randomly selected imperialist from GNDS. Assimilation procedure is provided in Algorithm 5.2.

Algorithm 5.2. Assimilation

1. Randomly select an imperialist I_G from GNDS.
2. **for each** colony in empire *i* **do**
3. set *d* to the element-wise difference of a colony and I_G
4. *Col_Pos_New = Col_Pos* + $\theta * \beta * \vec{r} \otimes \vec{d}$
5. **end for**
6. **if** rand() > p_e **do**
7. *R* = UpperBound – LowerBound;
8. **for each** decision variable *i* in *R* **do**
9. *w(i)* = (abs(*UpperBound*(i))*rand())$^{rand()/R(i)}$ – (abs(*LowerBound*(i))*rand())$^{(rand()/R(i)}$;
10. **end for**
11. *ColoniesOfEmpire* = *ColoniesOfEmpire* .* *w*;
12. **end if**

5.2.3 Revolution

The revolution operation in the proposed algorithm is completely different from the one in ICA because there is no random generation of new colonies. The new revolution operation has two parts, which are performed based on probability p_r. The first part is the generation of a new colony by the random selection of elements from two randomly selected imperialists in the LNDS set of the same empire. In the case

of having only one imperialist in the LNDS set, then one more individual is randomly generated. For the second part of the revolution process, some imperialists are modified and randomly chosen colonies are replaced with them. Revolution procedure is provided in Algorithm 5.3.

Algorithm 5.3. Revolution

1. **if** rand() > p_r
2. **for each** colony in empire *i* **do**
3. Select two imperialists $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$ from LNDS (if set contains one imperialist only, then generate one more randomly)
4. Generate new colony $\boldsymbol{C}$ by applying two-point crossover on $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$
5. Replace colony in an empire *i* with $\boldsymbol{C}$
6. **end for**
7. **else**
8. **for** *i*=1 to *RevolutionRate* * *NumberOfColoniesInEmpire*
9. Select one imperialist $\boldsymbol{I_i}$ from LNDS randomly
10. Update $\boldsymbol{I_i}$ by adding to its every element a random value between (0.001, 0.09) or (-0.09, -0.001)
11. **end for**
12. Update randomly selected colonies by newly generated ones
13. **end if**

Based on the preceding and present sections/subsections, it is evident that GNDS is used for selecting imperialists for updating colonies during the assimilation process. On the other hand, imperialists from LNDS of the same empire are used in the revolution process. Therefore, both assimilation and revolution of colonies enable the algorithm to escape local minima and reach global optimal solutions.

5.2.4 Possessing an Empire

Every empire is possessed by the set of imperialists, which is the non-dominated set of solutions within the empire itself. It is defined as LNDS in this algorithm. However, in terms of possession of the empire, it is possible that all individuals of an empire will be in the LNDS and thus there are no dominated solutions within an empire. As a result, assimilation and revolution will not be applicable in such a case. Therefore, one more parameter, ϕ, was added in this algorithm. It indicates the maximum percentage of imperialists that an empire can have. Consequently, when obtaining LNDS of an empire, the control of whether the percentage of imperialists exceeds ϕ in an empire is made. If it exceeds it, then the best maximum imperialists allowed are retained; the others are moved to the set of colonies. The total power of an empire is equal to the number of non-dominated solutions in the empire's population. Although an empire with a lower number of non-dominated solutions may contain better solutions than one with more non-dominated solutions, the total power is still equal to the cardinality measure regardless of the dominancy.

5.2.5 Uniting Similar Empires

MOICA uses different approaches in uniting similar empires with a comparison to the single objective version of it. This is because in a single objective algorithm, ICA, empires are united when each empire's imperialist is very close to the other's imperialist. This is achieved by calculating the distance between two positions of imperialists and comparing this calculated distance with the distance threshold parameter, which is originally set to 0.02. The distance threshold used here is not for diversity preservation; it is only used for measuring how close two empires are to each other. If the distance is less than or equal to the specified threshold, then the empires are united.

In the proposed algorithm, the mentioned approach for uniting similar empires cannot be applied because there are several imperialists in the empire. Thus, all imperialists must be considered to compare the empires for similarity. Consequently, a comparison of empires for similarity is implemented by using the generational distance metric [52], which enables the calculation of the generational distance between two or more sets of non-dominated solutions. The generational distance GD is defined as:

$$\mathrm{GD} = \frac{1}{|S^*|} \sum_{r \in S^*} \min\{d_{rx} \,|\, x \in S_j\} \tag{5.1}$$

where S^* is a reference solution set for evaluation of the solution set S_j and d_{xr} is the distance between a current solution x and a reference solution r as:

$$d_{rx} = \sqrt{\left(f_1(r) - f_1(x)\right)^2 + \left(f_2(r) - f_2(x)\right)^2 + \left(f_3(r) - f_3(x)\right)^2 + \ldots + \left(f_k(r) - f_k(x)\right)^2} \tag{5.2}$$

where k is the number of objective functions to be optimized. Reference and current solutions are the solutions from two empires to be united. Figure 5.4 illustrates an example for the computation of generational distance for two objective functions.

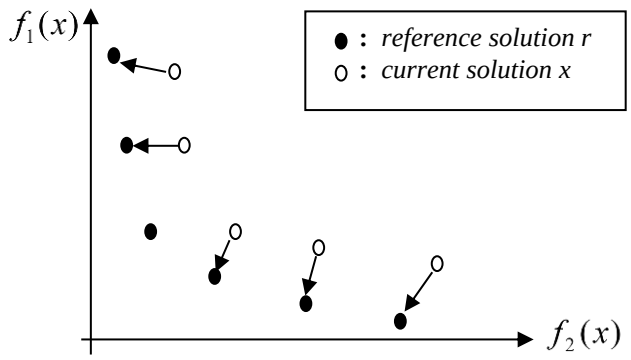


Figure 5.4. Generational distance for uniting empires

5.2.6 Imperialistic Competition

Imperialistic competition plays an important role in this algorithm because the whole algorithm is constructed to execute the competition between the empires. The imperialistic competition gradually decreases the number of weak empires, whereas it increases the number of strong empires. The weakest empire in the proposed algorithm is the one with the smallest number of non-dominated individuals, whereas the strongest empire is the one with the largest number of non-dominated individuals. Imperialistic competition is constructed so that the stronger an empire is, the more chances it has to obtain control of a weak colony in a weak empire. Consequently, it obtains possession of it. Weak empires will slowly lose their colonies during this competition and are soon terminated on account of being powerless, which means that these empires will be left with no countries. Imperialistic competition is described in Algorithm 5.4.

Algorithm 5.4. Imperialistic Competition

1. Construct a vector of total powers **P** for all empires.
2. Select the weakest empire **E** with the lowest total power.
3. Construct a vector of random values **R** ~$U(0,1)$ of size **P.**
4. Calculate **D** = **R - P** for each empire.
5. The empire with the maximum value in **D** will possess the randomly selected colony in empire **E.**
6. Terminate **E** if it has no colonies.

The complete flowchart of MOICA is presented in Figure 5.5.

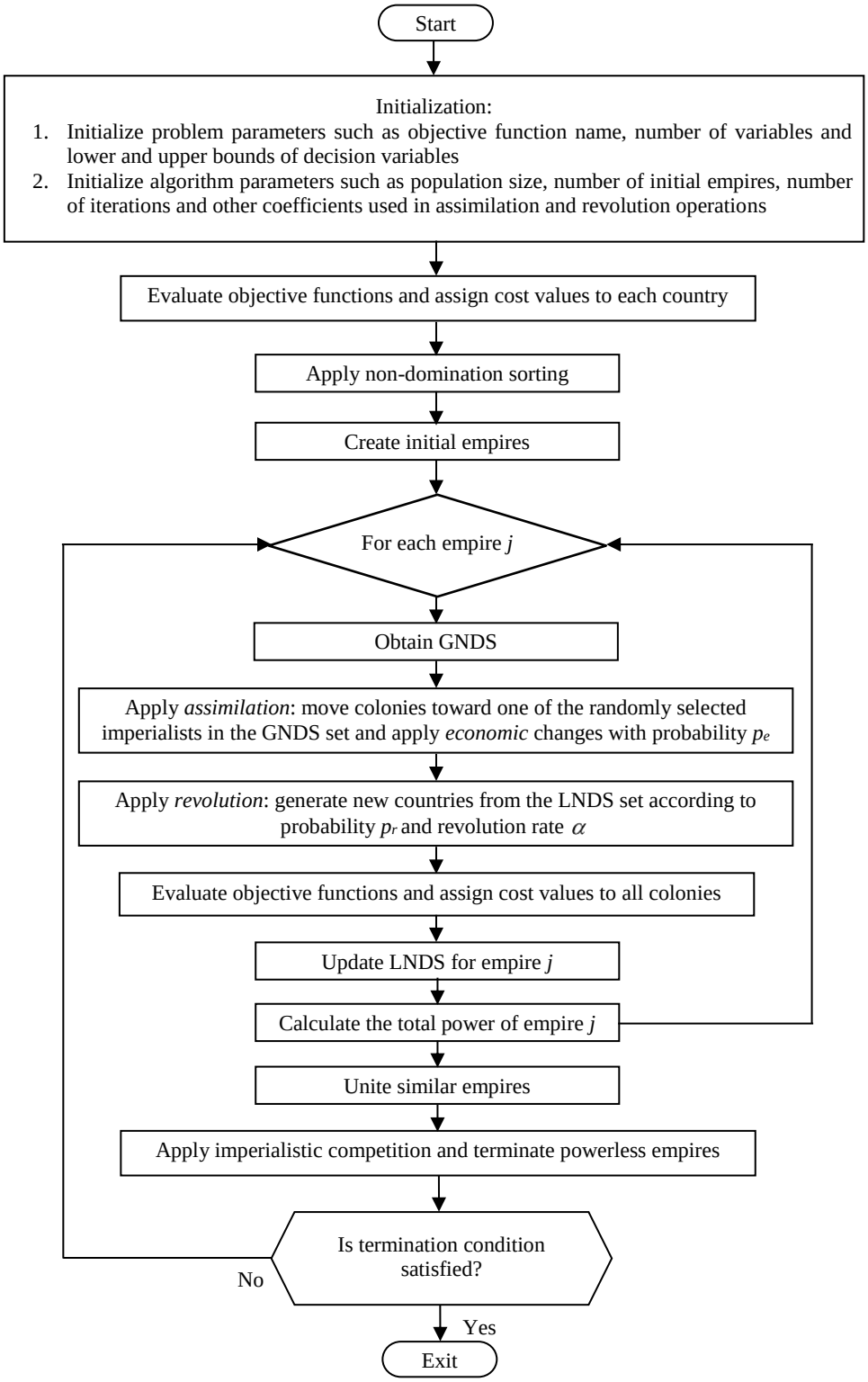


Figure 5.5. MOICA flowchart

5.2.7 Computational Complexity of MOICA

The time complexity for implementing non-domination sorting is the same as the time complexity for non-domination sorting in NSGA-II, that is $O(M(2N)^2)$, where M is the number of objectives and N is the number of solutions, i.e. the population size. Considering time complexities of assimilation and revolution operations, then in the worst case there is possibility for N-1 colonies to be assimilated / revolved, if there is only one dominating imperialist. Therefore, in every iteration, for both assimilation and revolution the time complexity is $O(N)$. Another part that is considered for time complexity is the uniting similar empires, where the time complexity in every iteration is $O(K^2)$, where K is the number of empires in the population. The time complexity of objective functions is $T(k)$, where k is the number of decision variables. As a result, the overall time complexity of MOICA is $O((M(2N)^2 + K^2) * T(k))$. Comparing the time complexities of MOICA and NSGA-II it can be concluded that they are almost the same, since K^2 is related to the number of empires, which is usually very low with comparison to the population size N, which could even be omitted.

5.3 Experimental Results

This section details the experiments and simulations conducted in this study. To obtain the experimental results and verify the effectiveness of the proposed algorithm, several bi- and tri-objective optimization problems were selected from the literature as the test problems. These were obtained from the study of Zitzler et al. [53]: ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6. Furthermore, test problems, including Kursawe [54], Fonseca [55], and Schaffer [56], were additionally used. Moreover, ten unconstrained test functions were employed from the Congress on Evolutionary

Competition (CEC) 2009 Special Session and Competition [57]: UF1, UF2, UF3, UF4, UF5, UF6, UF7, UF8, UF9, and UF10.

Table 5.1 details all unconstrained test problems used in this study except the CEC 2009 test functions, which can be found in [57]. Additionally, some performance metrics were used to evaluate the obtained results with the Pareto optimal solutions, specifically, *Hypervolume* (HV) [43], *Epsilon Indicator* (EI) [58] and *Inverted Generational Distance* (IGD). The IGD metric used in this study was the jMetal version.

In addition, this section presents a comparison of the results of the proposed algorithm and other state-of-the-art multi-objective optimization algorithms.

5.3.1 Discussion of the Results

All experimental results were obtained by executing each algorithm ten times. The maximum number of function evaluations was set to 25,000. For some test functions, it was set to 5,000 to verify the performance of the algorithms with a higher and lower number of function evaluations. The population size was set to 100 for all algorithms. The dimension of the individuals in the population was set to 30 for all test functions. The tables given below describe the average hyper-volume indicator as well as the epsilon indicator, which were obtained from several executions of the given algorithm. The IGD metric was obtained as the average value from several executions of the algorithms.

The proposed algorithm used the following parameters. The initial number of empires was set to eight. From several tests, it was evident that, if algorithm had far fewer or far more than eight initial empires, then the performance was poor. The

parameter θ had a random value between (0, 1) and β had a random value between (0, 5). The parameter for the percentage of imperialists ϕ was set to 0.3, such that at most 30% of the empire's population was considered imperialists. In addition, 70% of the space was left for colonies in an empire; thus, there were more assimilations and revolutions performed. The revolution rate α was set to 0.3, and the parameter used in the revolution process p_r was set to 0.5. On the other hand, the parameter for applying economic changes p_e may have been different for achieving better results in various test functions. For example, for UF9, the result was best when p_e was set to around 0.2; nonetheless, in most cases, it was between 0.8 and 1 based on a trial and error approach. The values for the above parameters were chosen as the best suitable values for the proposed algorithm after a number of conducted experiments. Therefore, the parameters for MOICA were tuned using a non-iterative algorithmic approach [59], such that the parameters were generated during initialization and were then tested.

The first three test problems addressed in this section are Fonseca, Kursawe, and Schaffer. Then, the ZDT set of problems is discussed and the results of the set of unconstrained problems in CEC 2009 are described. Values in bold are the best results obtained. All algorithms performed well in terms of convergence and divergence for each of the problems below. The cardinality measure, i.e. the number of non-dominated solutions, is important for having more candidate solutions and thus more chances for a good convergence. One of the main features that distinguish MOICA is the cardinality measure, which is very good for most of the problems.

The five real-valued ZDT problems are presented in Table 5.1, noting that ZDT5, the omitted problem, is binary encoded. Incidentally, due to being binary encoded, ZDT5 has often been omitted from analysis elsewhere in the EA literature.

Table 5.1. Unconstrained test problems used in this study

Problem	**Objective Functions**	**Variable bounds**	**n**
Fonseca	$f_1(x) = 1 - e^{-\sum_{i=1}^{n}(x_i - \frac{1}{\sqrt{n}})^2}$ $f_2(x) = 1 - e^{-\sum_{i=1}^{n}(x_i + \frac{1}{\sqrt{n}})^2}$	$-4 \le x_i \le 4$	3
Kursawe	$f_1(x) = \sum_{i=1}^{n-1}(-10e^{(-0.2*\sqrt{x_i^2 + x_{i+1}^2})})$ $f_2(x) = \sum_{i=1}^{n}(\lvert x_i \rvert^a + 5\sin x_i^b); a = 0.8; b = 3$	$-5 \le x_i \le 5$	3
Schaffer	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$-10 \le x \le 10^5$	1
ZDT1	$f_1(x) = x_1$ $f_2(x) = g(x)[1 - \sqrt{x_1 / g(x)}]$ $g(x) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \le x_i \le 1$	30
ZDT2	$f_1(x) = x_1$ $f_2(x) = g(x)[1 - (x_1 / g(x))^2]$ $g(x) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \le x_i \le 1$	30
ZDT3	$f_1(x) = x_1$ $f_2(x) = g(x)[1 - \sqrt{\frac{x_i}{g(x)}} - \frac{x_1}{g(x)}\sin(10\pi x_1)]$ $g(x) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \le x_i \le 1$	30
ZDT4	$f_1(x) = x_1$ $f_2(x) = g(x)[1 - (x_1 / g(x))^2]$ $g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}[x_i^2 - 10\cos(4\pi x_i)]$	$0 \le x_1 \le 1$ $-5 \le x_i \le 5$ $i = 2,\ldots,n$	30
ZDT6	$f_1(x) = 1 - e^{-4x_1}\sin^6(6\pi x_1)$ $f_2(x) = g(x)[1 - (f_1(x)/g(x))^2]$ $g(x) = 1 + 9[(\sum_{i=2}^{n} x_i)/(n-1)]^{0.25}$	$0 \le x_i \le 1$	30

Tables 5.2 to 5.4 contain *Hypervolume*, *Epsilon Indicator*, and *IGD* results of MOICA, NSGA-II, SPEA2 and OMOPSO for the unconstrained test problems given

in Table 5.1. On average, results for *Hypervolume* and *Epsilon Indicator* are similar for all algorithms. However, MOICA performs much better in terms of *IGD* metric.

Table 5.2. *Hypervolume* results for unconstrained test problems with 25,000 function evaluations

Algorithm / Fun.	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
Fonseca	0.99441	0.99441	0.99447	**0.99453**
Kursawe	**1.00000**	**1.00000**	**1.00000**	**1.00000**
Schaffer	**0.97764**	0.81099	0.90619	0.97748
ZDT1	0.99242	0.99713	0.99699	**0.99725**
ZDT2	0.98534	0.99431	0.99392	**0.99444**
ZDT3	0.99824	0.99833	**0.99835**	0.99831
ZDT4	**0.98440**	0.72624	0.57710	0.02393
ZDT6	**0.97120**	0.93140	0.90948	0.97105

Table 5.3. *Epsilon Indicator* results for unconstrained test problems with 25,000 function evaluations

Algorithm / Fun.	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
Fonseca	1.00470	1.00560	1.00520	**1.00310**
Kursawe	**1.04330**	1.04890	1.04890	1.04620
Schaffer	1.01020	1.01030	1.07750	**1.01010**
ZDT1	1.03240	1.03330	1.03580	**1.01360**
ZDT2	1.00350	1.00348	1.00329	**1.00250**
ZDT3	**1.00000**	0.99970	0.99950	0.99910
ZDT4	**1.03260**	3.43550	5.38910	22.01040
ZDT6	1.01610	1.31910	1.47070	**1.01590**

Table 5.4. *IGD* results for unconstrained test problems with 25,000 function evaluations

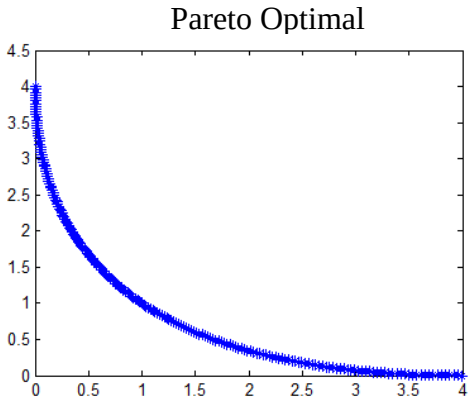
Algorithm / Fun.	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
Fonseca	**1.1805E-4**	3.2267E-4	2.3427E-4	2.1033E-4
Kursawe	4.4896E-4	1.7598E-4	**1.3464E-4**	1.6165E-4
Schaffer	**2.3575E-5**	0.0373	0.0215	3.3629E-4
ZDT1	**2.5732E-5**	1.8641E-4	1.5222E-4	1.3782E-4
ZDT2	**3.5707E-5**	1.9656E-4	1.7261E-4	1.4183E-4
ZDT3	**7.4842E-5**	2.6488E-4	2.3718E-4	2.1859E-4
ZDT4	**3.8724E-5**	0.0849	0.1365	1.1501
ZDT6	**1.6200E-5**	0.0137	0.0219	1.2514E-4

Table 5.5 contains MOICA, MOEA/D-AWA [60], Harmony NSGA-II and Harmony MOEAD [61] IGD results for the set of ZDT problems. From these results it is seen that MOICA outperforms all three algorithms.

Table 5.5. *IGD* results for ZDT test problems with 25,000 function evaluations

Algorithm / **Fun.**	***MOICA***	***MOEA/D-AWA***	***Harmony NSGA-II***	***Harmony MOEAD***
ZDT1	**2.5732E-5**	4.470e–3	8.03e-04	1.86e-03
ZDT2	**3.5707E-5**	4.482e–3	1.12e-03	3.01e-03
ZDT3	**7.4842E-5**	6.703e–3	5.01e-04	1.19e-03
ZDT4	**3.8724E-5**	4.238e–3	8.33e-02	1.64e-04
ZDT6	**1.6200E-5**	4.323e–3	2.11e-04	1.91e-04

As stated above, one of the features of MOICA is the ability to produce many candidate solutions by having a high cardinality measure. The Schaffer test problem is an example for illustrating the cardinality measure of MOICA.



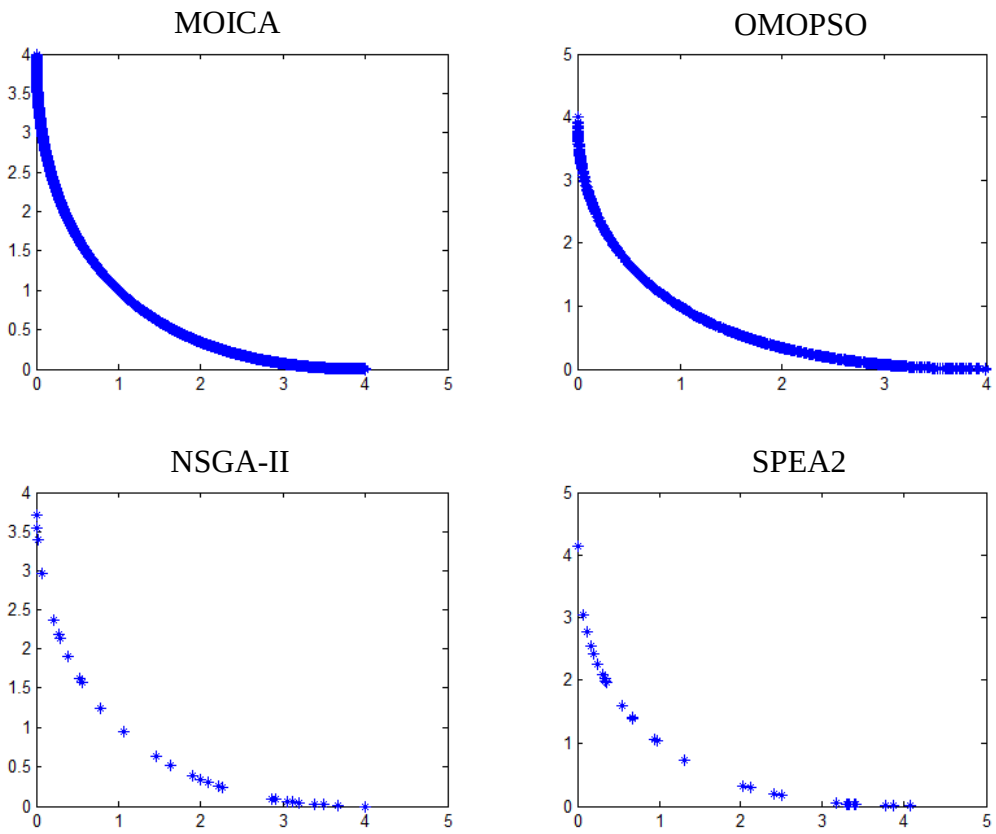


Figure 5.6. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on Schaffer

Figure 5.6 illustrates the Pareto found by all algorithms for the Schaffer test problem. Although the *Hypervolume* and *Epsilon Indicator* results are good for all algorithms, as shown in Tables 5.2 and 5.3, respectively, it is easily seen from Figure 5.6 that MOICA and OMOPSO have much better cardinality measures than NSGA-II and SPEA2.

For ZDT1, ZDT2, and ZDT3 test problems, all algorithms performed equally well. However, with respect to the ZDT4 test problem, MOICA performed much better than all other algorithms in this study. In the ZDT4 test problem, MOICA demonstrated its power in terms of convergence and divergence when compared to the other algorithms. It was successful in this test problem and others on account of

the way in which it searches the available search space. This is handled by setting many different empires in the beginning of the algorithm for which LNDS sets are positioned in different parts of the search space. This enables the algorithm to search the whole search space and to consequently obtain good convergence and divergence. Figure 5.7 illustrates the Pareto found by four algorithms for the ZDT4 test problem, which demonstrates how the spread of solutions, convergence, and divergence is effectively preserved in MOICA compared to other algorithms. Figure 5.8 illustrates ZDT6 test problem, which is another good example for illustrating the performance of MOICA compared to other algorithms used in this study. However, in ZDT6, both MOICA and OMOPSO performed well compared to NSGA-II and SPEA2; however, NSGA-II performed better than SPEA2.

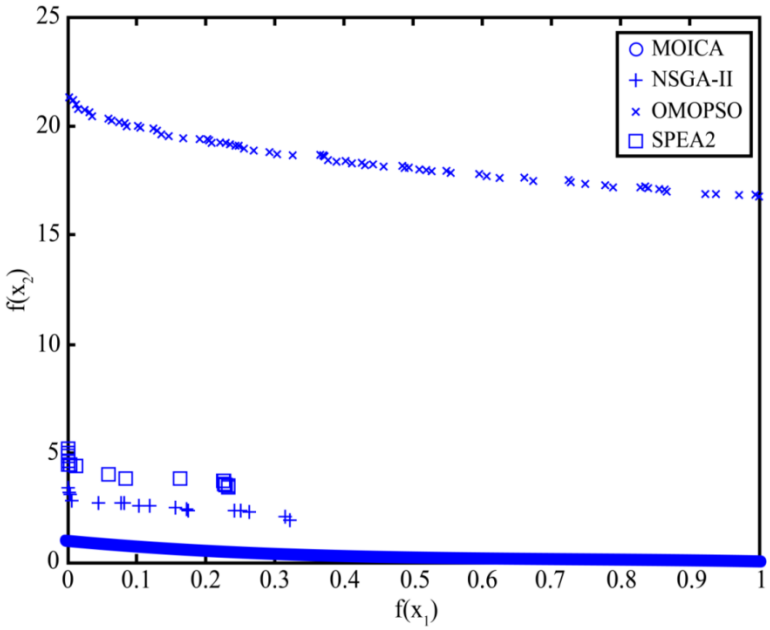


Figure 5.7. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on ZDT4

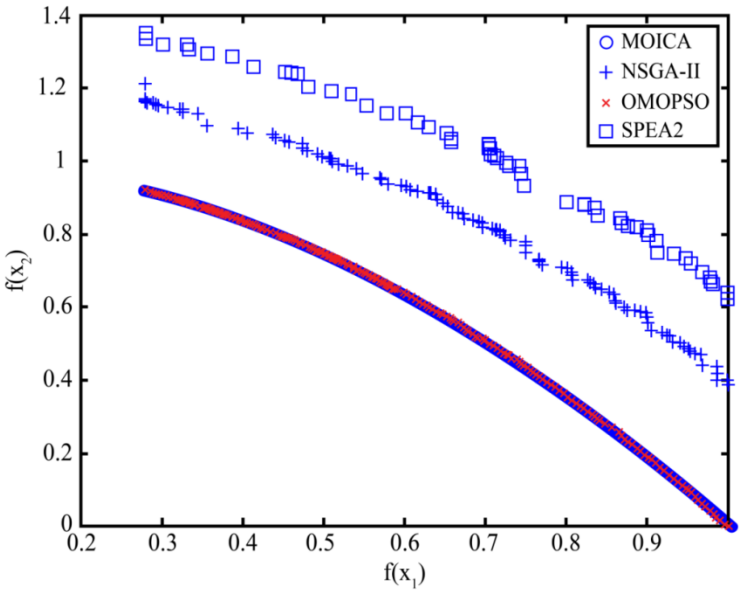


Figure 5.8. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on ZDT6

Meanwhile, Tables 5.6 to 5.9 contain *Hypervolume, Epsilon Indicator, and IGD* results for Uf1-UF10 unconstrained test problems from CEC 2009 with 25,000 function evaluations for which MOICA on average again produces reasonably good results.

Table 5.6. *Hypervolume* results for CEC 2009 unconstrained test problems with 25,000 function evaluations

Algorithm / **Fun.**	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
UF1	0.98701	0.97802	0.98667	**0.98955**
UF2	**0.99671**	0.98934	0.98897	0.99279
UF3	0.92884	0.94478	0.98728	**0.99454**
UF4	**0.98849**	0.98838	0.98811	0.98728
UF5	**0.93230**	0.91786	0.90060	0.81056
UF6	0.93785	**0.94459**	0.94282	0.91297
UF7	0.97662	0.96890	0.95315	**0.98730**
UF8	**0.99348**	0.99280	0.99220	0.98945
UF9	**0.98309**	0.97468	0.95914	0.97845
UF10	**0.99334**	0.92076	0.92967	0.72864

Table 5.7. *Epsilon Indicator* results for CEC 2009 unconstrained test problems with 25,000 function evaluations

Algorithm / Fun.	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
UF1	**1.07890**	1.11220	1.06800	2.03310
UF2	1.14590	**1.12880**	1.13060	1.22760
UF3	1.70320	1.83810	**1.07410**	1.50760
UF4	**1.06600**	1.08930	1.07390	1.07410
UF5	**1.55180**	1.81000	1.86610	6.19530
UF6	1.62380	1.56080	**1.42700**	2.22700
UF7	1.07390	**1.03760**	1.05000	1.57840
UF8	**2.04970**	4.01980	2.91510	7.51780
UF9	4.19050	6.75860	**3.61460**	23.76640
UF10	**1.23930**	3.25450	2.96740	7.65290

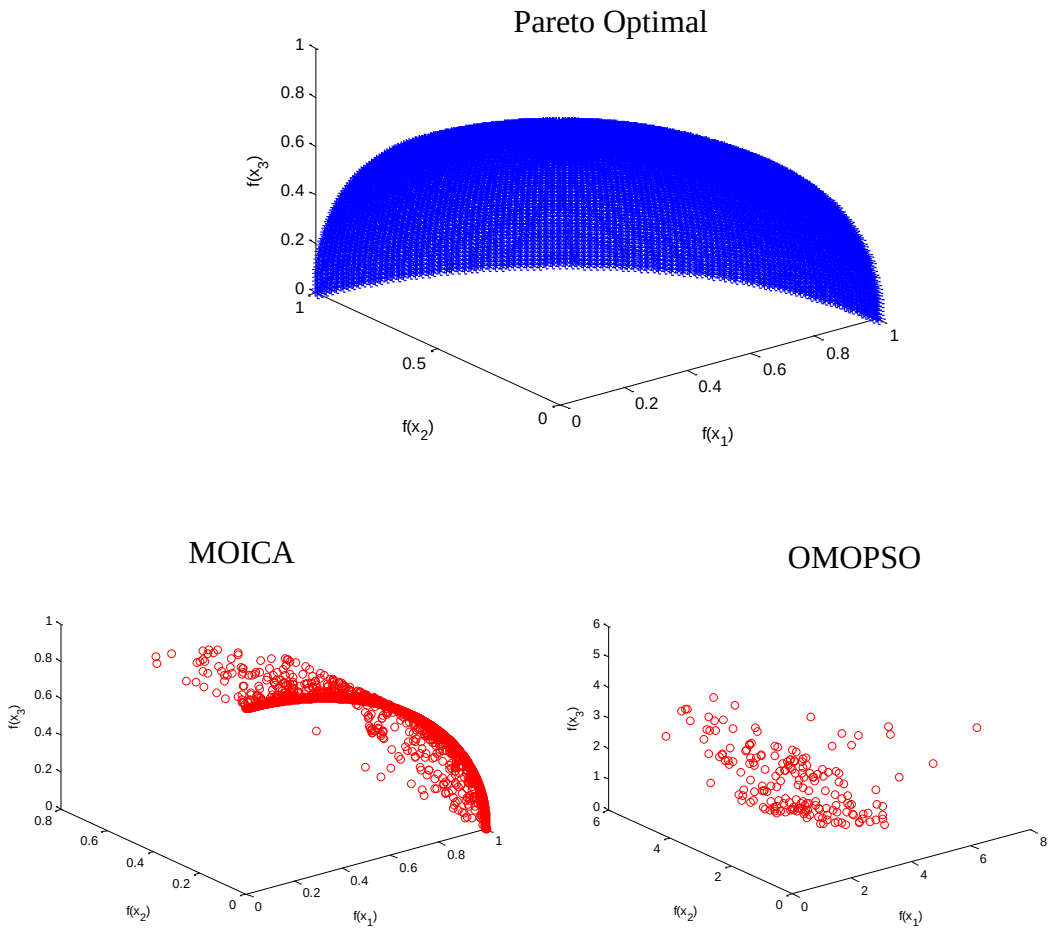
Table 5.8. *IGD* results for CEC 2009 unconstrained test problems with 25,000 function evaluations

Algorithm / Fun.	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
UF1	**0.0035**	0.0047	0.0042	0.0041
UF2	**0.0018**	0.0020	0.0024	0.0021
UF3	0.0103	0.0084	0.0072	0.0072
UF4	**0.0018**	**0.0018**	0.0019	0.0022
UF5	0.1268	**0.1117**	0.1155	0.3579
UF6	0.0127	**0.0102**	0.0119	0.0178
UF7	0.0075	0.0068	0.0098	**0.0036**
UF8	**0.0026**	0.0029	0.0027	0.0037
UF9	0.0035	0.0035	**0.0030**	0.0056
UF10	**0.0037**	0.0063	0.0046	0.0266

Table 5.9. *IGD* results for CEC 2009 unconstrained test problems with 25,000 function evaluations

Algorithm / Fun.	***MOICA***	***DMCMOABC***	***Harmony NSGA-II***	***Harmony MOEAD***
UF1	**0.0035**	0.0053	0.0037	0.0026
UF2	**0.0018**	0.0050	0.0345	**0.0018**
UF3	0.0103	0.0544	0.0085	**0.0067**
UF4	**0.0018**	0.0254	0.0033	0.0021
UF5	0.1268	0.0527	**0.0457**	0.0488
UF6	0.0127	0.0393	**0.0089**	0.0092
UF7	0.0075	**0.0065**	0.0113	0.0118
UF8	**0.0026**	0.0665	0.0028	0.0054
UF9	0.0035	0.0368	0.0044	0.0060
UF10	**0.0037**	0.1119	**0.0036**	0.0059

The performance of MOICA with regard to the test functions in CEC 2009 is good because MOICA produced competitive results compared to the other algorithms. On the other side DMCMOABC performs worst while Harmony NSGA-II and Harmony MOEAD perform also well. Figure 5.9 presents the results of MOICA along with those of other algorithms, as well as the Pareto optimal for the UF10 unconstrained test function. Tables 5.6 to 5.9 illustrate that MOICA performs better than the other algorithms with respect to UF10. In addition, it is clear from Figure 5.9 that MOICA is within the objective space of the Pareto optimal, unlike the other algorithms.



NSGA-II SPEA2

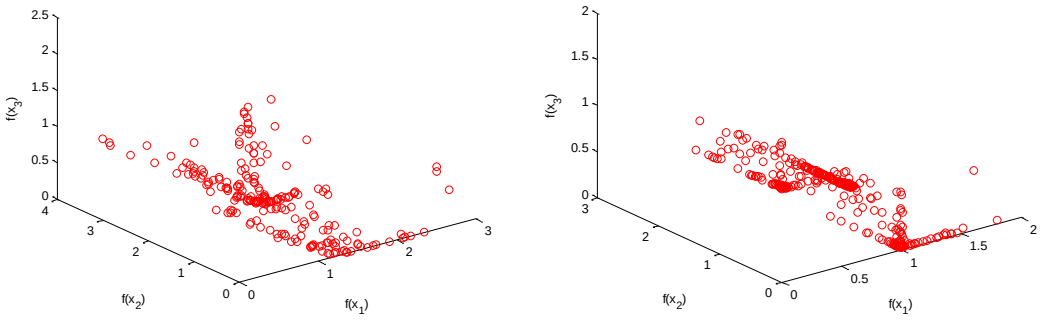


Figure 5.9. Non-dominated solutions of MOICA, OMOPSO, NSGA-II and SPEA2 on UF10

Tables 5.10 to 5.12 show the results for UF1-UF7 test functions with a maximum of 5,000 function evaluations. The average performance of MOICA is either similar or better than the performances of the other algorithms, even for such a low number of function evaluations. This result likewise proves that MOICA quickly converges to global optimal solutions.

Table 5.10. *Hypervolume* results for CEC 2009 unconstrained test problems with 5,000 function evaluations

Algorithm / Function	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
UF1	0.97192	0.97651	**0.98519**	0.97075
UF2	0.98314	0.98818	0.98617	**0.98854**
UF3	0.91334	0.90554	0.89972	**0.97219**
UF4	**0.98567**	0.98377	0.98167	0.98521
UF5	**0.86472**	0.83347	0.80558	0.74621
UF6	**0.88661**	0.88456	0.88426	0.86270
UF7	0.97156	0.94940	0.93761	**0.97825**

Table 5.11. *Epsilon Indicator* results for CEC 2009 unconstrained test problems with 5,000 function evaluations

Algorithm / Fun.	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
UF1	**1.22630**	1.57160	1.60980	1.45710

Table 5.11 (continued)

UF2	1.51470	1.31100	**1.27830**	1.35280
UF3	2.82790	2.37100	2.50600	**1.61580**
UF4	1.12490	1.14260	1.13340	**1.11650**
UF5	**2.87590**	4.36320	2.98440	5.24710
UF6	2.45680	**2.32760**	3.01710	5.07890
UF7	**1.12350**	1.23610	1.44560	1.43810

Table 5.12. *IGD* results for CEC 2009 unconstrained test problems with 5,000 function evaluations

Algorithm / Fun.	***MOICA***	***NSGA-II***	***SPEA2***	***OMOPSO***
UF1	0.0054	**0.0045**	0.0051	0.0064
UF2	0.0033	0.0032	0.0033	**0.0030**
UF3	0.0152	0.0156	0.0153	0.0101
UF4	**0.0027**	0.0031	0.0033	**0.0027**
UF5	**0.2167**	0.3346	0.3409	0.5060
UF6	**0.0229**	0.0264	0.0243	0.0357
UF7	**0.0061**	0.0089	0.0117	0.0066

Table 5.13 contains comparison of results between MOICA, MOEP with rank sorting, MOEP with non-domination sorting [64] and MOSaDE algorithms [88]. This comparison also proves very good performance of MOICA, since it produces better results than others.

Table 5.13. *IGD* results for CEC 2009 unconstrained test problems with 25,000 function evaluations

Algorithm / Fun.	***MOICA***	**MOEP** (rank sorting)	**MOEP** (non-domination sorting)	**MOSaDE**
UF1	**0.0035**	0.0596	0.0588	0.0983
UF2	**0.0018**	0.0189	0.0516	0.0607
UF3	**0.0103**	0.0990	0.1910	0.3248
UF4	**0.0018**	0.0427	0.0624	0.0977
UF5	**0.1268**	0.2245	0.7608	0.6963
UF6	**0.0127**	0.1031	0.3606	0.3640
UF7	**0.0075**	0.0197	0.0408	0.1916
UF8	**0.0026**	0.4230	0.6512	0.4019
UF9	**0.0035**	0.3420	0.2744	0.3984
UF10	**0.0037**	0.3621	2.4987	2.9313

Table 5.14 presents the ranking of MOICA compared to the algorithms used in the CEC 2009 competition for unconstrained functions. The ranking is based on the average IGD metric.

Table 5.14. Ranking of MOICA compared to algorithms in CEC 2009

UF1	IGD	UF2	IGD	UF3	IGD
MOICA	0.0035	MOICA	0.0018	MOEAD	0.00742
MOEAD	0.00435	MTS	0.00615	MOICA	0.0103
GDE3	0.00534	MOEADGM	0.0064	LiuLi Algorithm	0.01497
MOEADGM	0.0062	DMOEADD	0.00679	DMOEADD	0.03337
MTS	0.00646	MOEAD	0.00679	MOEADGM	0.049
LiuLi Algorithm	0.00785	OWMOSaDE	0.0081	MTS	0.0531
DMOEADD	0.01038	GDE3	0.01195	Clustering MOEA	0.0549
NSGAIILS	0.01153	LiuLi Algorithm	0.0123	AMGA	0.06998
OWMOSaDE	0.0122	NSGAIILS	0.01237	DECMOSA-SQP	0.0935
Clustering MOEA	0.0299	AMGA	0.01623	MOEP	0.099
AMGA	0.03588	MOEP	0.0189	OWMOSaDE	0.103
MOEP	0.0596	Clustering MOEA	0.0228	NSGAIILS	0.10603
DECMOSA-SQP	0.07702	DECMOSA-SQP	0.02834	GDE3	0.10639
OMOEAII	0.08564	OMOEAII	0.03057	OMOEAII	0.27141
UF4	IGD	UF5	IGD	UF6	IGD
MOICA	0.0018	MTS	0.01489	MOEAD	0.00587
MTS	0.02356	GDE3	0.03928	MOICA	0.0127
GDE3	0.0265	AMGA	0.09405	MTS	0.05917
DECMOSA-SQP	0.03392	MOICA	0.1268	DMOEADD	0.06673
AMGA	0.04062	LiuLi Algorithm	0.16186	OMOEAII	0.07338
DMOEADD	0.04268	DECMOSA-SQP	0.16713	Clustering MOEA	0.0871
MOEP	0.0427	OMOEAII	0.1692	MOEP	0.1031
LiuLi Algorithm	0.0435	MOEAD	0.18071	DECMOSA-SQP	0.12604
OMOEAII	0.04624	MOEP	0.2245	AMGA	0.12942
MOEADGM	0.0476	Clustering MOEA	0.2473	LiuLi Algorithm	0.17555
OWMOSaDE	0.0513	DMOEADD	0.31454	OWMOSaDE	0.1918
NSGAIILS	0.0584	OWMOSaDE	0.4303	GDE3	0.25091

Table 5.14 (continued)

Clustering MOEA	0.0585	NSGAIILS	0.5657	NSGAIILS	0.31032
MOEAD	0.06385	MOEADGM	1.7919	MOEADGM	0.5563
UF7	IGD	UF8	IGD	UF9	IGD
MOEAD	0.00444	MOICA	0.0026	MOICA	0.0035
LiuLi Algorithm	0.0073	MOEAD	0.0584	DMOEADD	0.04896
MOICA	0.0075	DMOEADD	0.06841	NSGAIILS	0.0719
MOEADGM	0.0076	LiuLi Algorithm	0.08235	MOEAD	0.07896
DMOEADD	0.01032	NSGAIILS	0.0863	GDE3	0.08248
MOEP	0.0197	OWMOSaDE	0.0945	LiuLi Algorithm	0.09391
NSGAIILS	0.02132	MTS	0.11251	OWMOSaDE	0.0983
Clustering MOEA	0.0223	AMGA	0.17125	MTS	0.11442
DECMOSA-SQP	0.02416	OMOEAII	0.192	DECMOSA-SQP	0.14111
GDE3	0.02522	DECMOSA-SQP	0.21583	MOEADGM	0.1878
OMOEAII	0.03354	Clustering MOEA	0.2383	AMGA	0.18861
MTS	0.04079	MOEADGM	0.2446	OMOEAII	0.23179
AMGA	0.05707	GDE3	0.24855	Clustering MOEA	0.2934
OWMOSaDE	0.0585	MOEP	0.423	MOEP	0.342
UF10	IGD				
MOICA	0.0037				
MTS	0.15306				
DMOEADD	0.32211				
AMGA	0.32418				
MOEP	0.3621				
DECMOSA-SQP	0.36985				
Clustering MOEA	0.4111				
GDE3	0.43326				
LiuLi Algorithm	0.44691				
MOEAD	0.47415				
MOEADGM	0.5646				
OMOEAII	0.62754				
OWMOSaDE	0.743				
NSGAIILS	0.84468				

5.3.2 Friedman Aligned Ranks Test

In addition to the ranking based on average IGD results discussed in previous section a Friedman aligned ranks test is also implemented over all IGD scores in order to compare statistical similarities between results of MOICA and other 13 algorithms in CEC2009 MOO contest. Table 5.15 shows the average rank values for all algorithms along with the p-value. The best scores of algorithms are shown by the subscripted numbers in Table 5.15. It is clear from Table 5.15 that, the average rank value of MOICA is the smallest one, which indicates that MOICA is the best performing algorithm among the 14 competitors. Meanwhile, the p-value being very small implies that there is significant statistical difference among the results as well as MOICA being statistically different from other algorithms.

Table 5.15. Friedman aligned ranks over CEC2009 UF problem instances

Algorithm	***Average F_{AR} value***	***p-value***
MOEAD	$4{,}45_{(4)}$	1.6544e-06
GDE3	$7{,}50_{(6)}$	
MOEADGM	$8{,}50_{(8)}$	
MTS	$4{,}4_{(3)}$	
LiuLi Algorithm	$5{,}90_{(5)}$	
DMOEADD	$4{,}35_{(2)}$	
NSGAIILS	$9{,}50_{(10)}$	
OWMOSaDE	$9{,}90_{(12)}$	
Clustering MOEA	$9{,}60_{(11)}$	
AMGA	$8{,}30_{(7)}$	
MOEP	$9{,}50_{(10)}$	
DECMOSA-SQP	$8{,}70_{(9)}$	
OMOEAII	$10{,}60_{(13)}$	
MOICA	$1{,}70_{(1)}$	

Chapter 6

CONCLUSION

This thesis presents studies and research on the single-objective and multi-objective optimization algorithms. Imperialistic competitive algorithm, which is a single-objective global optimization algorithm, was studied in details and also applied on NP-hard problem – Travelling Salesman Problem. In addition to this, ICA was improved by modifying it's the most important operator – assimilation, and as a result a new algorithm ICAMA was developed. Finally, a multi-objective version of ICA - MOICA was proposed and applied on various real valued benchmark problems known in the literature. The novelty of MOICA is highlighted in having several non-dominated solution sets, where exploration and exploitation of a search space is done without using special parameter for diversity preservation, which enables algorithm to avoid extra computations for maintaining spread of solutions. Both, ICAMA and MOICA performed well with comparison to other state-of-the-art algorithms. Experiments conducted in this thesis illustrate how especially MOICA outperforms other algorithms for most of the test problems. Application of Fuzzy Logic with ICAMA or MOICA is an option for the future work. There are many problems in real world that need to be solved by multi-objective optimization algorithms including fuzziness in their approach.

REFERENCES

[1] Sherinov, Z., Unveren, A., & Acan, A. (2011). An Evolutionary Multi-Objective Modeling and Solution Approach for Fuzzy Vehicle Routing Problem. *INISTA 2011, IEEE Conference*.

[2] Atashpaz-Gargari, E., & Lucas, C. (2007). Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. *IEEE Congress on Evolutionary Computation*.

[3] Melanie M. (1999). An Introduction to Genetic Algorithms. Massachusetts, MIT Press, pp. 2-12.

[4] Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, vol. 344, no. 2–3, pp. 243–278.

[5] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, pp. 1942–1948.

[6] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, vol. 220, no. 4598, pp. 671–680.

[7] Wang, B., Guan, Z., Li, D., Zhang, C., & Chen, L. (2014). Two-sided assembly line balancing with operator number and task constraints: a hybrid imperialist competitive algorithm. *Int J Adv Manuf Technol*, vol. 74, no. 5–8, pp. 791–805.

[8] Ramezanian, R. & Ezzatpanah, A. (2015). Modeling and solving multi-objective mixed-model assembly line balancing and worker assignment problem. *Computers & Industrial Engineering*, vol. 87, pp. 74–80.

[9] Lian, K., Zhang, C. Z., Gao, L., & Xinyu, S. (2011). Single row facility layout problem using an imperialist competitive algorithm, *Proc. of the 41st Int. Conf. on Computers and Industrial Engineering*, pp. 578–586.

[10] Hosseini, S., Khaled, A. A., & Vadlamani, S. (2014). Hybrid imperialist competitive algorithm, variable neighborhood search, and simulated annealing for dynamic facility layout problem. *Neural Computing and Applications*, vol. 25, no. 7–8, pp. 1871–1885.

[11] Xu, S., Wang, Y., & Huang, A. (2014). Application of Imperialist Competitive Algorithm on Solving the Traveling Salesman Problem. *Algorithms*, vol. 7, no. 2, pp. 229–242.

[12] Hosseini, S. M., Khaled, A. A., & Jin, M. (2012). Solving Euclidean minimal spanning tree problem using a new meta-heuristic approach: Imperialist Competitive algorithm (ICA). *2012 IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 176-181.

[13] Mortazavi, A., Khamseh, A. A., & Naderi, B. (2015). A novel chaotic imperialist competitive algorithm for production and air transportation scheduling problems. *Neural Computing and Applications*, vol. 26, no. 7, pp. 1709–1723.

[14] Agha Mohammad Ali Kermani, M., Aliahmadi, A., Salamat, V. R., Barzinpour, F., & Hadiyan, E. (2014). Supplier selection in a single-echelon supply chain with horizontal competition using Imperialist competitive algorithm. *International Journal of Computer Integrated Manufacturing*, vol. 28, no. 6, pp. 628–638.

[15] Talebi, H., Nikoo, H., & Mirzaei, A. (2012). Face verification in complex background based on imperialist competitive algorithm. *The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012)*.

[16] Moghaddam, M. E. & Nemati, N. (2013). A robust color image watermarking technique using modified Imperialist Competitive Algorithm. *Forensic Science International*, vol. 233, no. 1–3, pp. 193–200.

[17] Abdechiri, M., Faez, K., & Bahrami, H. (2010). Neural Network Learning Based on Chaotic Imperialist Competitive Algorithm. *2010 2nd International Workshop on Intelligent Systems and Applications*.

[18] Duan, H. & Huang, L. (2014). Imperialist competitive algorithm optimized artificial neural networks for UCAV global path planning. *Neurocomputing*, vol. 125, pp. 166–171.

[19] Mousavirad, S. J. & Ebrahimpour-Komleh, H. (2013). Feature selection using modified imperialist competitive algorithm. *ICCKE 2013*.

[20] Mahmoodabadi, Z. & Shaerbaf Tabrizi, S. (2014). A New ICA-Based Algorithm for Diagnosis of Coronary Artery Disease. *Intelligent Computing, Communication and Devices*, pp. 415–427.

[21] Roche, R., Idoumghar, L., Blunier, B., & Miraoui, A. (2012). Imperialist Competitive Algorithm for Dynamic Optimization of Economic Dispatch in Power Systems. *Artificial Evolution*, pp. 217–228.

[22] Mehdinejad, M., Mohammadi-Ivatloo, B., Dadashzadeh-Bonab, R., & Zare, K. (2016). Solution of optimal reactive power dispatch of power systems using hybrid particle swarm optimization and imperialist competitive algorithms. *International Journal of Electrical Power & Energy Systems*, vol. 83, pp. 104–116.

[23] Seidgar, H., Kiani, M., Abedi, M., & Fazlollahtabar, H. (2013). An efficient imperialist competitive algorithm for scheduling in the two-stage assembly flow shop problem. *International Journal of Production Research*, vol. 52, no. 4, pp. 1240–1256.

[24] Ghasemishabankareh, B., Shahsavari-Pour, N., Basiri, M.-A., & Li, X. (2016). A Hybrid Imperialist Competitive Algorithm for the Flexible Job Shop Problem. *Artificial Life and Computational Intelligence*, pp. 221–233.

[25] Bagher, M., Zandieh, M., & Farsijani, H. (2010). Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm. *The International*

Journal of Advanced Manufacturing Technology, vol. 54, no. 1–4, pp. 271–285.

[26] Talatahari, S., Farahmand Azar, B., Sheikholeslami, R., & Gandomi, A. H. (2012). Imperialist competitive algorithm combined with chaos for global optimization. *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 3, pp. 1312–1319.

[27] Niknam, T., Taherian Fard, E., Pourjafarian, N., & Rousta, A. (2011). An efficient hybrid algorithm based on modified imperialist competitive algorithm and K-means for data clustering. *Engineering Applications of Artificial Intelligence*, vol. 24, no. 2, pp. 306–317.

[28] Razmjooy, N., Mousavi, B. S., & Soleymani, F. (2013). A hybrid neural network Imperialist Competitive Algorithm for skin color segmentation. *Mathematical and Computer Modelling*, vol. 57, no. 3–4, pp. 848–856.

[29] Roozbeh Nia, A., Hemmati, M. Far, & Niaki, S. T. A. (2015). A hybrid genetic and imperialist competitive algorithm for green vendor managed inventory of multi-item multi-constraint EOQ model under shortage. *Applied Soft Computing*, vol. 30, pp. 353–364.

[30] Ray, S. S., Bandyopadhyay, S., & Pal, S. K. (2005). New Genetic Operators for Solving TSP: Application to Microarray Gene Ordering. *PReMI 2005, LNCS* 3776, pp. 617–622.

[31] Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102.

[32] Suganthan, P.N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. -P., Auger, A., & Tiwari, S. (2005). Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, *Technical Report*.

[33] Chen, Q., Lin, B., Zhang, Q., Liang, J. J., Suganthan, P. N., & Qu, B. Y. (2015). Problem Definitions and Evaluation Criteria for CEC 2015 Special Session on Bound Constrained Single-Objective Computationally Expensive Numerical Optimization, *Technical Report*.

[34] Karaboga, D. & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471.

[35] Price, K., Storn, R. M., & Lampinen, J. A. (2005). Differential Evolution: A Practical Approach to Global Optimization. *Springer-Verlag Berlin Heidelberg*.

[36] Schwefel, H. P. (1995). Evolution and Optimum Seeking. *Wiley Interscience*, New York.

[37] Derrac, J., García, S., Molina, D., & Herrera, F. (2011). 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, no. 1, pp. 3–18.

[38] Reeves, C. (2002). Genetic Algorithms. *Principles and Perspectives*, Vol. 20, Springer.

[39] Deb, K. (2001). Multiobjective optimization using evolutionary algorithms. *Wiley, Chichester*, UK.

[40] Fonseca, C.M., & Fleming, P.J. (1993). Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: *Forrest (ed) Proc 5th Int Conf Genetic Algor, Morgan Kauffman*, S. San Mateo, CA, pp 416–423.

[41] Horn, J., Nafploitis, N., & Goldberg, D.E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. In: *Michalewicz Z (ed) Proc 1st IEEE Conf Evolut Comput, IEEE Press*, Piscataway, NJ, pp 82–87.

[42] Srinivas, N., & Deb, K. (1995). Multiobjective function optimization using nondominated sorting genetic algorithms. *Evolut Comput* (2)3:221–248.

[43] Zitzler, E., & Thiele, L. (1998). Multiobjective optimization using evolutionary algorithms: a comparative case study. In: *Eiben et al. (eds) Parallel problem solving from nature, Springer-Verlag*, Berlin, pp 292–301.

[44] Seyedmohsen, H., & Abdullah, A.K. (2014). A survey on the imperialist competitive algorithm metaheuristic: implementation in engineering domain and directions for future research. *Appl Soft Comput* 24:1078–1094.

[45] Duan, H., Xu, C., Liu, S., & Shao, S. (2010). Template matching using chaotic imperialist competitive algorithm. *Pattern Recogn Lett* 31:1868–1875.

[46] Nazari-Shirkouhi, S., Eivazy, H., Ghodsi, R., Rezaie, K., & Atashpaz-Gargari, E. (2010). Solving the integrated product mix-outsourcing problem by a novel meta-heuristic algorithm: imperialist competitive algorithm. *Expert Syst Appl* 37:7615–7626.

[47] Vedadi, M., Vahidi, B., & Hosseinian, S. H. (2015). An Imperialist Competitive Algorithm Maximum Power Point Tracker for photovoltaic string operating under partially shaded conditions. *Sci.Int.(Lahore)*, 27(5), 4023-4033.

[48] Goudarzi, M., Vahidi, B., & Naghizadeh, R. A. (2013). Optimum reactive power compensation in distribution networks using Imperialistic Competitive Algorithm. *Sci. Int. (Lahore)*, 25(1), 27-31.

[49] Kashani, A. R., Gandomi, A. H., & Mousavi, M. (2014). Imperialistic Competitive Algorithm: A metaheuristic algorithm for locating the critical slip surface in 2-Dimensional soil slopes. *Geoscience Frontiers*, 1-7.

[50] Jordehi, A. R. (2016). Optimal allocation of FACTS devices for static security enhancement in power systems via imperialistic competitive algorithm (ICA). *Applied Soft Computing* 48, 317–328.

[51] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans Evolut Comput* (6)2:182–197.

[52] Van Veldhuizen, D.A., & Lamont, G.B. (1998). Multiobjective evolutionary algorithm research: a history and analysis. *Dept Elect Comput Eng, Grad School Eng, Air Force Inst. Technol, Wright-Patterson AFB, OH, Tech. Rep.* TR-98-03.

[53] Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: empirical results. *Evolut Comput* (8)2:173–195.

[54] Kursawe, F. (1990). A variant of evolution strategies for vector optimization. In: Schwefel H-P and Manner R (eds) Parallel problem solving from nature. *Springer-Verlag, Berlin*, pp 193–197.

[55] Fonseca, C.M., & Fleming, P.J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms—part II: application example. *IEEE Trans Syst Man Cybern* (A) 28:38–47.

[56] Schaffer, J.D. (1987). Multiple objective optimization with vector evaluated genetic algorithms. In: *Grefensttete (ed) Proc 1st Int Conf Genetic Algor*, Lawrence Erlbaum, Hillsdale, NJ, pp 93–100.

[57] Zhang, Q., Zhou, A., Zhao, S., Suganthan, P.N., Liu, W., & Tiwari, S. (2009). Multiobjective optimization test instances for the CEC 2009 Special Session and Competition. *Tech Report CES-487*, University of Essex, Essex, UK.

[58] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Grunert da Fonseca, V. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Trans Evolut Comput* 7(2):117–132.

[59] Eiben, A.E., & Smit, S.K. (2011). Evolutionary algorithm parameters and methods to tune them. In: *Hamadi et al. (eds) Autonomous search*. Springer, Berlin, pp 15–36.

[60] Qi, Y., Ma, X., Liu, F., Jiao, L., Sun, J., & Wu, J. (2014). MOEA/D with adaptive weight adjustment. *Evolutionary computation*, 22(2), 231-264.

[61] Doush, I. A., & Bataineh, M. Q. (2015). Hybedrized NSGA-II and MOEA/D with Harmony Search Algorithm to Solve Multi-objective Optimization Problems. In: Arik S., Huang T., Lai W., Liu Q. (eds) *Neural Information Processing*. Lecture Notes in Computer Science, vol 9489. Springer, Cham.

[62] Ebrahimzadeh, A., Addeh, J., & Rahmani, Z. (2012). Control chart pattern recognition using K-MICA clustering and neural networks. *ISA Trans*. 51 (1) 111–119.

[63] Sherinov, Z., & Unveren A. (2013). ICA for solving Travelling Salesman Problem. *Seventh International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control (ICSCCW)*, 2-4.

[64] Qu, B. Y., & Suganthan, P. N. (2009). Multi-objective evolutionary programming without non-domination sorting is up to twenty times faster". In: *Proceeding of Congress on Evolutionary Computation*, CEC '09, 2934-2939.

[65] Huang, V.L., Zhao, S.Z., Mallipeddi, R., Suganthan, P.N. (2009). Multi-objective Optimization Using Self-adaptive Differential Evolution Algorithm. In: *Proc. CEC*, pp. 190-194. IEEE, Los Alamitos.

[66] Zitzler, E., Laumanns, M., & Thiele, L. (2002). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In: *Proc. Evol. Methods Design, Optimization Control Applicat. Ind. Problems (EUROGEN '01)*, Barcelona, Spain: Inte. Center Numerical Methods Eng. (CIMNE), pp. 95–100.

[67] Zhang, Q. & Li, H. (2007). MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6), pp. 712-731.

[68] Rueda, J.L., & Erlich, I. (2015). MVMO for Bound Constrained Single-Objective Computationally Expensive Numerical Optimization. *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1011-1017.

[69] Tanweer, M. R., Suresh, S., & Sundararajan, N. (2015). Improved SRPSO algorithm for solving CEC 2015 computationally expensive numerical optimization problems. *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1943-1949.

[70] Al-Dujaili, A., Subramanian, K., & Suresh, S. (2015). Humancog: A cognitive architecture for solving optimization problems. *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 3220-3227.

[71] Hansen, N., Müller, S.D., Koumoutsakos, P. (2003). Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES), *Evolutionary Computation*, 11(1), pp. 1-18.

[72] Andersson, M., Bandaru, S., Ng, A., & Syberfeldt, A. (2015). Parameter tuned CMA-ES on the CEC'15 expensive problems. *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1950-1957.

[73] Xiang, Y., & Zhou, Y. (2015). A dynamic multi-colony artificial bee colony algorithm for multi-objective optimization. *Appl Soft Comput* 35:766–785.

[74] Zhang, Q. F., Liu, W. D., & Li, H. (2009). The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances. In: *Proceedings of*

the IEEE Congress on Evolutionary Computation (CEC '09), pp. 203–208, IEEE.

[75] Kukkonen, S. & Lampinen, J. (2007). Performance assessment of Generalized Differential Evolution 3 (GDE3) with a given set of problems. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '07)*, pp. 3593–3600.

[76] Chen, C.-M., Chen, Y.-P., & Zhang, Q. F. (2009). Enhancing MOEA/D with guided mutation and priority update for multi-objective optimization. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 209–216.

[77] Tseng, L.-Y. & Chen, C. (2009). Multiple trajectory search for unconstrained/constrained multi-objective optimization. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 1951–1958.

[78] Liu, H.-L. & Li, X. Q. (2009). The multiobjective evolutionary algorithm based on determined weight and sub-regional search. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 1928–1934.

[79] Liu, M. H., Zou, X. F., Yu, C., & Wu, Z. J. (2009). Performance assessment of DMOEA-DD with CEC 2009 MOEA competition test instances. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 2913–2918, IEEE, Trondheim, Norway.

[80] Sindhya, K., Sinha, A., Deb, K., & Miettinen, K. (2009). Local search based evolutionary multi-objective optimization algorithm for constrained and unconstrained problems. Proc. IEEE CEC, pp. 2919-2926.

[81] Wang, Y. P., Dang, C. Y., Li, H. C., Han, L. X., & Wei, J. X. (2009). A clustering multi-objective evolutionary algorithm based on orthogonal and uniform design. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 2927–2933, IEEE, Trondheim, Norway.

[82] Tiwari, S., Fadel, G., Koch, P., & Deb, K. (2009). Performance assessment of the hybrid archive-based micro genetic algorithm (AMGA) on the CEC09 test problems. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 1935–1942, IEEE, Trondheim, Norway.

[83] Zamuda, A., Brest, J., Bošković, B., & Žumer, V. (2009). Differential evolution with self-adaptation and local search for Constrained multiobjective optimization. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 195–202.

[84] Gao, S., Sanyou, Xiao, B., Zhang, L., Shi, Y., Tian, X., Yang, Y., Long, H., Yang, X., Yu, D., & Yan, Z. (2009). An orthogonal multi-objective evolutionary algorithm with lower-dimensional crossover. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '09)*, pp. 1959–1964.

[85] Reyes Sierra, M. & Coello Coello, C. A. (2005). Improving PSO-Based Multi-objective Optimization Using Crowding, Mutation and ϙ-Dominance. *In*

Evolutionary Multi-Criterion Optimization (EMO 2005), LNCS 3410, pp. 505–519.

[86] Deb, K., Agrawal, S., Pratab, A., Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 681-695.

[87] Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the Performance of the Strength Pareto Evolutionary Algorithm. *Technical Report 103, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH)*, Zurich.

[88] Huang, V. L., Qin, A. K., Suganthan, P. N., & Tasgetiren, M. F. (2007). Multi-objective optimization based on self-adaptive differential evolution algorithm. *Proc. Int. Conf. IEEE Congr. Evol. Comput.*, pp. 3601-2608.

[89] Geem, Z.W., Kim, J.H., & Loganathan, G.V. (2001). A new heuristic optimization algorithm: harmony search. *Simulation* 76(2), 60–68.

[90] Zitzler, E. & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and and the strength pareto approach. *IEEE Trans, on Evolutionary Computation*, 3(4):257-271.

[91] Zou, X., Chen, Y., Liu, M. & Kang, L. (2008). A New Evolutionary Algorithm for Solving Many-objective Optimization Problems. *IEEE Trans. on System, Man and Cybernetics*, Part B, vol.38, pp.1402-1412.

[92] Zou, X., Liu, M., Kang, L., & He, J. (2004). A high performance multi-objective evolutionary algorithm based on the principles of thermodynamics. In: *Proc. Parallel Problem Solving from Nature 8th Int. Conf.*, vol.3242, LNCS, X. Yao, E. Burke, J.A. Lozano, J. Smith, J.J. Merelo-Guervós, J.A. Bullinaria, J. Rowe, P. Tino, A. Kabán, and H.-P.Schwefel, Eds., Berlin, Germany: Springer-Verlag, pp.922-931.

[93] Qin, A. K. & Suganthan, P. N. (2005). Self-adaptive differential evolution algorithm for numerical optimization. In: *IEEE Congress on Evolutionary Computation (CEC 2005)* Edinburgh, Scotland, IEEE Press, pp. 1785-1791.

[94] Zamuda, A., Brest, J., Boškovíc, B., & Žumer, V. (2008). Študija samoprilagajanja krmilnih parametrov pri algoritmu DEMOwSA. *Elektrotehniška vestnik*, vol.75, no.4, pp. 223-228.

[95] Sherinov, Z., Ünveren, A. & Acan, A. (2017). Imperialist Competitive Algorithm with Updated Assimilation for the Solution of Real Valued Optimization Problems. *International Journal on Artificial Intelligence Tools*.

[96] Sherinov, Z. & Ünveren, A. (2017). Multi-objective imperialistic competitive algorithm with multiple non-dominated sets for the solution of global optimization problems. *Soft Computing*, Springer.